Growing Grounds Inventory Application

College of Computer Engineering

Senior Project

June 2013

By: Courtney Schenkl

Advisor: Lynne Slivovsky

Table of Contents

1. Background Information

Project Overview

The Growing Grounds application aims to make the process of counting bloomed flowers simpler while also formatting data in a digital form so that buyers can more easily see what they are buying through the application's features. The applications main feature will be its fast and easy generation of the nursery's weekly availability lists through an integrated database API.

Project Goal

The goal of this application is to simplify the inventory taking process that is currently in place at the Growing Grounds nursery. By having the current process broken into multiple stages of making changes on a physical paper, and then transferring that collected data into an excel spreadsheet that is then saved and printed out for buyers to use the process takes an estimated six hours every week. This large amount of counting time will be drastically cut by moving all of this information onto a digital database that can be manipulated and changed on a weekly basis through the Android application.

Objectives

The objectives listed below are broken down into miniature milestones that are necessary to be met in order for the completion of a working mobile application to be delivered to Growing Grounds in June.

- Develop a SQLite database with the Growing Grounds full plant catalog to be stored in the app.
- Implement a dynamically changing database system with the proper functionality to generate the weekly availability lists through proper queries.
- Allow for the integration of the camera API into the application so that QR codes can be scanned and used to represent the plant's botanical and common names for ease of use.

2. Outcomes and Deliverables

I plan on building a working prototype of a Growing Grounds Android Application that will operate on an Android tablet. This application is meant to replace the current use of excel spreadsheets and paper copies, that then have to be translated back into the excel spreadsheet, with one medium for the recording of plant availability and generation of availability lists for the buyers. The hope for this prototype is that it can be used on any Android based tablet in any weather conditions, the use of the application will drastically cut down the current counting time (6 hours) and the system will be easy to use by any employee or buyer interested in the nursery's plant availability.

Duration

Task	Task Name	Start Date	End Date	Duration
1	Make GUI Skeleton	3/1/2013	3/8/2013	1 W
2	Load the Full Catalog into App's Database	3/9/2013	3/12/2013	3 D
Break For Finals Week Studying and Test Taking		**3/18/2013**	**3/22/2013**	**1 W**
3	Implement Database's Query Functionality	3/22/2013	4/6/2013	2 W 1D
4	Test that Querying generates proper output (availability list)	4/7/2013	4/14/2013	1 W
5	Add Querying GUI to make availability list	4/14/2013	4/21/2013	1 W
6	Add auto-complete functionality for plant names in database	4/21/2013	4/28/2013	1 W
7	Add feature to export the list via email	4/28/2013	5/5/2013	1 W
8	Add feature for adding pictures to each database entry (plant)	5/5/2013	5/12/2013	1 W
9	Test the full functionality of the application.	**Remainder of Quarter Until 6/7/2013**		

3. Planning Information

A work breakdown structure that identifies the work to be done to complete the project would be necessary for a better project plan. The work breakdown structure will be used to separate the design of the Android application into important tasks or “major milestones” that need to be completed incrementally for the successful development of the project. The resources that are being utilized for this project are just labor, the Android SDK, and an Android tablet. A future tablet should be purchased for Growing Grounds use after the development process is completed.

This project will be developed with the engineering design process in mind. The steps to this process are as follows: define the problem, doing background research, specifying requirements, brainstorming solutions, choosing the best solution, developing the product, building a prototype, testing and redesign. Since the use of a database system is the straightforward design that has been agreed upon between the client (Growing Grounds) and development team (myself and my advisor Lynne Slivosky), it takes care of researching implementation methods, and brainstorming solutions to the current proposed needs of the client. The steps of development, building (which can be thought of as implementing the application’s software), testing and redesign will still be utilized and will be very important to successful completion of the Growing Grounds Application.

The main constraint that has not been summarized, but isn’t a concern in the development of the application, is the budget for a new tablet that will be given to Growing Grounds for their use of the completed application.

Marketing Requirements

1. Easily Updatable
2. Lightweight
3. Compact
4. Fast
5. Waterproof
6. Weatherproof (sun, wind, ect)
7. Long Battery Life

Engineering Requirements

The engineering and respective marketing requirements, along with justification are listed in the below table:

Marketing Requirement	**Engineering Requirement**	**Justification**
1	The application should be easily updated through the Google Play market place (if the app is put into the market) or through simple steps with specified instructions of reloading an updated program onto the tablet	The two options are more dependent on how much time in the quarter remains for loading the app onto the Google Play marketplace, but both yield the same result of easily updating the application so that bugs can be fixed, or new features can be added if new demands arise.
2	The weight of the entire system being used to facilitate the application should be a maximum of ¼ lb heavier than the tablet's weight.	This has to do with the fact that whoever is using the application is going to be walking around the entire nursery keeping track of the blooming plants. Since this process takes around 2-3 hours (for an experienced employee) the equipment that we are using cannot be too heavy as to make the user's arms ache or get tired from carrying around the extra weight.
3	The application should only require the tablet that is being used and a holding harness for the user's hand to fit in to balance the tablet while they are using the application.	This has to do with user comfort. Users are working for a long amount of time, and the equipment that they are using must be conducive for them to get the work done quickly without having to balance a large piece of equipment.

4	This application must provide instant feedback from the stored database in order to generate the weekly availability sheets. Querying should also provide for instant feedback to the user. There shouldn't be any lag time between the user's commands and the results that come of that command.	This has to be done so that the employee can get the work done as quickly as possible with the aid of this app. On the buyer's end of the spectrum, the app has to perform very quickly as well so that the buyer can see as much information as possible within the short amount of time they have to purchase different plants and flowers.
5	Must fit into IP52BW waterproof standards	This standard says that water molecules entering the system will not cause it to break and that the system can withstand a steady droplet (like rain) flow of water without malfunctioning. This is needed because the job has to be done in any weather situation. Rain or sleet or snow.
6	The tablet must have a screen protector attached to it, and also provide Growing Grounds with extra protectors and instructions for application.	Smudges from fingers and exposure to sun could end up causing damage to a touch screen. In order to prevent that damage we will be using screen protectors so that the damages can be prevented.
7	Battery must be able to last for up to 3 hours with the application running.	This has to account for the tablet being able to have enough charge to complete the task of generating each weeks availability logs. This has been shown to take anywhere from 2-4 hours.

Criteria

- Processor Speed (GHz) → the faster is more desirable
- Weight (lbs.) → the lighter is more desirable
- Battery Life (hours) → the longer is more desireable

4. The User Experience

Overview

The users of this application fall into three categories: an experienced plant counter, and inexperienced counter and a buyer. The following personas focus on generic people who fall into those three categories and how easy/difficult and helpful they would find the end product of this application.

Personas

Persona #1

Background:

Name: Jane

Age: 26

Jane is a new hire at Growing Grounds and she is technically inept. She has multiple devices at home (tablet, smartphone, notebook, desktop) and is in tuned with the current trends in technology. She has no real experience with the nursery, and has trouble navigating through the different parts of the grounds.

Behavior Patterns:

- ➢ Is always on her phone checking different social media sites, making phone calls, sending texts…ect.
- ➢ Has trouble getting set in new environments when it comes to working
- ➢ Is very friendly and can get along with many people. Doesn't have a problem asking people for help.

Goals:

- ➢ Wants to get a better grasp on where everything is located within the Growing Grounds nursery.
- ➢ Loves the idea of using a mobile application to do the work of generating weekly availability charts and wants to use it.
- ➢ Wants to master application system of collecting data and teach the rest of the staff, since she if very "in tuned" with newer forms of technology.

Skills:

- ➢ Extremely computer/technologically literate.

- Quick at learning new tasks.
- Is a “people” person and loves communicating and being around other people.

Attitudes:

- Very open to new ideas and ways of doing things
- Loves being around people and is considered to be a happy person
- Always using any type of technology and is up on all of the “hip” applications

Environment:

- Comes from upper middle class economic standing.
- Grew up doing community service for similar organizations.
- Was always raised knowing that she should always help people in whatever way possible; whether through educating people on things she had a greater grasp on, or though community service work.

Blog Title: Day 1 of the New Job

Today I started working at Growing Grounds and was put on the duty of generating the weekly availability charts. This week wasn’t too bad because I just shadowed the person who usually does the job. The thing that worried me about the entire job was the fact that I have no idea where anything is! My work friend, who is a master at this job, was able to see a flower/plant and just automatically know which part of this HUGE nursery it was at; I don’t think I will EVER be able to do this. Shoot….I don’t want to get fired!

Blog Title: NEW APP!!!

After doing some research on how to learn how to recognize plants more easily (so I don’t look like a lost puppy navigating the nursery) and this new tablet application popped up that was made for Growing Grounds. I asked my supervisor if we were using it and she said yes. That was my green light to nerd out and figure out this app. I downloaded it on my tablet and saw that the nursery location was on each flower/plant’s information and I almost did a backflip I was so happy! I mastered how to use it and on the next Monday when we were making the availability sheet I felt like I now was able to do the job and was able to even help other people learn how to use the app!

Persona #2

Background:

Name: Jack

Age 55

Jack was one of the original volunteers that worked at Growing Grounds. He knows the nursery like the back of his hand and loves working with the people and plants that are encapsulated within Growing Grounds. Jack doesn't own a cell phone and is "old school" in his beliefs on technology. Jack has a desktop computer at home but other than that, he has no other forms of technology and has no want or desire to own or use any of those devices.

Behavior Patterns

- Jack is an extremely hard worker and goes the extra mile to help everyone out with all of their tasks if they are having problems.
- Jack knows the nursery and its set up like the back of his hand.

Goals:

- He is in charge of training people with generating the availability charts in the paper and excel combination system that has always been in place.
- He has seen the younger employees using the app, and has seen them taking less time than he (a master) takes with the old system. He now wants to learn how to use this new application.

Skills:

- Is very well versed in different horticultural areas and can make the connection between a plant and its latin based name very easily.
- Has been in the organization for a long time and knows how to do all of the tasks in the quickest and most efficient ways.

Attitudes:

- Resistant to change
- Resistant to technology
- Likes to get work done as quickly as possible
- Is willing to learn tasks if they are proven to be the most efficient way of doing something.

Environment:

Jack knows everything about Growing Grounds and has mastered every single job that there is to do at the nursery. He has recently seen the newer volunteers using a tablet to generate the weekly availability sheets in a faster amount of time than his current system and is willing to learn how to use the tablet and the app if it means that the sheets can be generated faster.

Blog Title: These Kids and Their Toys

I have noticed that some of the newer volunteers, the younger kids, are using a tablet to make our weekly availability sheets. I was totally against using this system until I saw that these new people (who have no idea where any of the plants are located) were consistently finishing with their availability sheets than I was. Because of this I decided to figure out how to use this tablet to do the work.

The tablet was pretty easy to figure out because the set up was very simple and straight forward. I thought it was really cool how there were also pictures included in each entry of the table. Now the buyers will have a better idea of what the plants will look like at different points of their life. And on a side note, I figured out how the kids got the work done so easy without knowing where anything was, the app tells them where the plant is located….cheaters.

Persona #3

Background

Name: Drew

Age: 35

Drew works for a mainstream company that has a floral section in their stores. They have been buying from Growing Grounds for the past couple of years and are very pleased with how their plants bloom for their customers. Drew is in charge of going out to the Growing Grounds nursery and inspecting the plants that are ready to be sold and then making the purchasing decisions for the company he works for.

Behavior Patterns:

- Is good with different forms of technology.
- Owns a laptop, smart phone and also a tablet and can use them with ease.
- Is always busy and can usually only stays in one place for a half an hour max.

Goals:

- Wants to spend less time at the Growing Grounds nursery because it is rather out of the way from his other daily appointments.
- Would love it if the Growing Grounds representatives could email him availability charts so that he could ask more people's opinions at his company as to which plants they should buy.

Skills:

- People person and very friendly.
- Very good with using newer technological devices.

Attitudes:

- Constantly busy and is always on the move
- Faster he can be in and out of a meeting the better
- Loves talking and being around people but has to stick to a very strict schedule

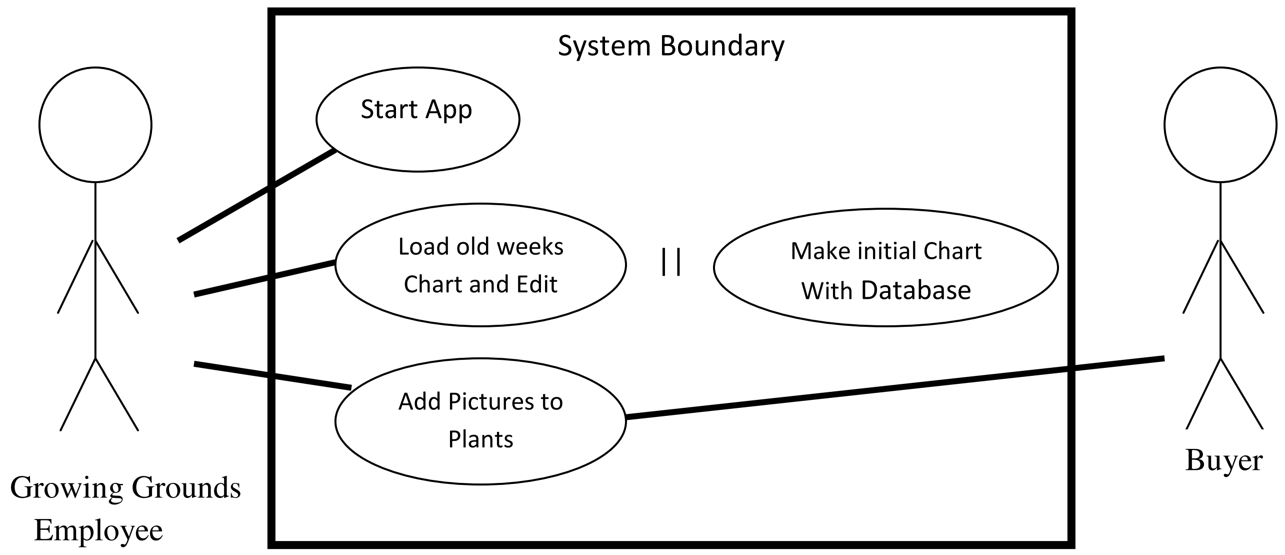
Environment:

Drew absolutely loves Growing Grounds. He loves how nice the people are, how beautiful the plants are and how reliable the company is to always produce gorgeous plants. Even though he loves all of those things about the company, they also are some of the things he hates because it kills his schedule. The people are so friendly and always want to talk about life and catch up, but Drew just doesn't have the time. He also has to rely on a trained eye to see if a plant in the back of their truck (filled with flowers/plants for sale) will look amazing when it hits his company's shelves.

Blog Title: Whoa! That was FAST!

After going to Growing Grounds for the past two years I have never been in and out of a buying appointment as quickly as I was today! They are using a new application on an android tablet to show me their buying charts. It is AMAZING! I can look through it very quickly by searching plant names or whether or not they are in bloom and results come up on the page! They also included pictures of each of the plants throughout their life time, from bud to bloom so I can see what they will look like at all points in their lifetime. THIS IS AMAZING! So much easier and so much faster!

Use-Cases



- Title: Client Using System
- Main Success Scenario:
    - Start up the app on the android tablet.
    - Either pull up the precious weeks chart to make edits to it, or make the initial chart by querying the main database.
    - Add pictures to flowers/plants based on their current state
    - The buyer can then view the chart on the tablet and see all of the pictures to make their buying decisions.

5. Project Planning

Resources

- Human:
  - Courtney Schenkl
  - Dr. Slivovsky
  - Megan Hall
  - Craig Wilson
- Technical:
  - Development Environment:
    - Eclipse
    - Android Emulator
  - Equipment:
    - Samsung GALAXY Tab 2 – 10.1
  - Physical:
    - Access to the Growing Grounds Nursery for testing

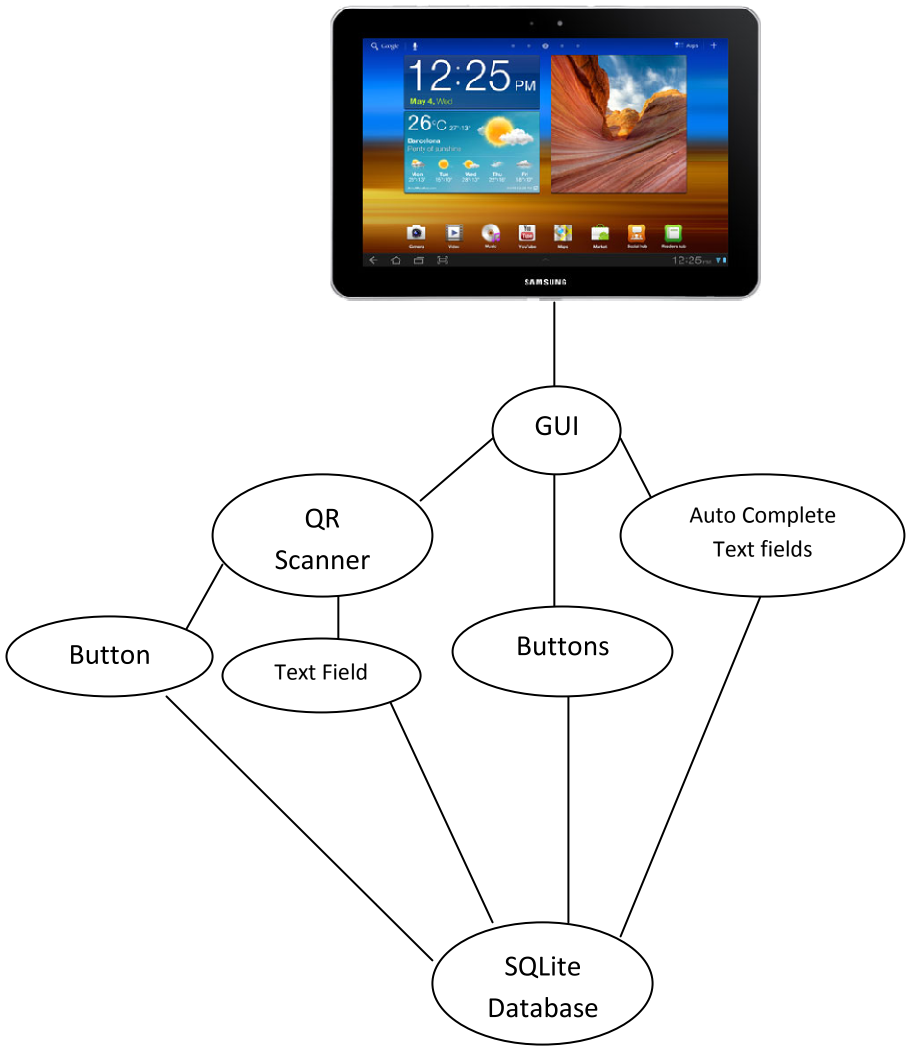
Milestones, Artifacts, Dependencies and Schedule
(This is all based off of the in depth detailed schedule seen in the Outcomes and Deliverables Section)

Start Date	End Date	Milestone	Artifact
3/1/13	3/8/13	GUI Skeleton	GUI Skeleton
3/8/13	4/6/13	Implementation of the Databases querying functionality should be completed.	GUI with ability to make queries and show results.
4/6/13	4/28/13	App should have functionality added for auto-complete for plant names, and displaying the generated. availability list.	All of the features listed can be seen through the application.
4/28/13	5/12/13	Added features of exporting generated list to an email to be printed and saved to main computers, and added feature of being able to	All of the features listed will be seen thorugh the application.

		store pictures in the database entries as well.	
5/12/13	6/7/13	Making changes to GUI and functionality that are needed after running field tests.	The completed application.

6. Design and Justification

a. System Architecture



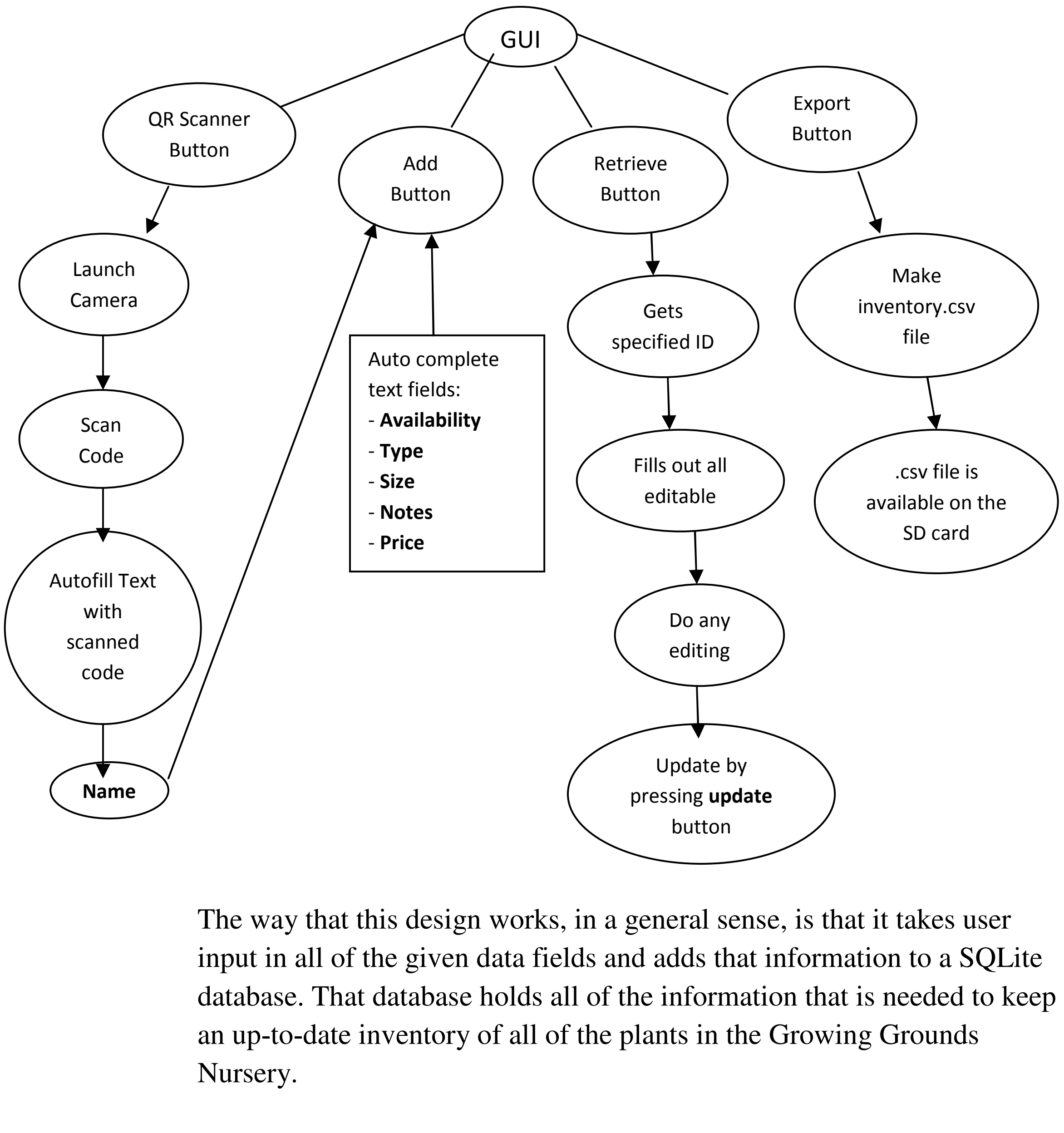
The way that this diagram is set up is to show the basic functionality of the system. The only hardware component that is being used is the Android Galaxy 10.1 Tab, and the rest of the system is composed of the software. The software is split up between the GUI (graphical user interface) and the actual backbone software, which makes up the SQLite database. The GUI is composed of buttons and text fields.

The two types of text fields that are editable are the auto complete fields that the user has to physically touch and type input into (for Availability, Type, Size, Notes and Price), and then there is just a normal text field that is paired with a "Scan" button that, when pressed, will launch the tablet's camera to scan a code and then fill in the text that is to its left with the scanned information.

To add information into the database, the "add" button is pressed and the information is inputted. The same is true for editing, except instead of re-inputting all of the data, the user can pull up the entry of choice to edit and the text fields to edit will be filled with the information that is currently in the database.

b. Software Architecture

i. Design Diagram



The way that this design works, in a general sense, is that it takes user input in all of the given data fields and adds that information to a SQLite database. That database holds all of the information that is needed to keep an up-to-date inventory of all of the plants in the Growing Grounds Nursery.

The first input field is the QR scanner button. When this button is pressed, the application will launch the tablet's camera with the QR scanner window open. This window gives a small line of instructions as to how scan a QR code ("Line up QR code with the rectangle to scan"), and has the actual scanner present. Once you have successfully scanned in a plant name the application will be redirected to the main page except the text field that used to direct the user to click the scan button will now be filled with the contents that the QR scan provided. This is how you get the "name" field for the database entry.

To obtain the other five entries that are needed to make a row in the inventory list the user will press each of the labeled text fields and fill them out based on the guidelines given above them. For example, the "Type" field has the short guideline of "(press space bar for options)," and when the user does that, all of the possible types will appear in a drop down menu to aid in them being able to choose the proper type with ease. The rest of the fields have the same capability of auto completing the users input after they begin typing the first character.

After all of these fields have been filled out and the QR scan has been done, the user can press the "Add" button and that row will be added into the inventory database.

If there was a mistake made when entering in the values for a particular row, the user can retrieve that row of data and edit it. To do this, they will fill out the "id" field with the row id that they wish to retrieve and then the database will be queried to bring the details of that row into the text fields to the left of the retrieve button. After the user has edited the fields to their satisfaction they can hit the update button which will re-add that row into the database with all of the corrections.

After the user has finished inputting the entire database, they can choose to export that information into a ".csv" file. This file type is a comma delimited file that allows for the information that was taken on the tablet to be transferred into an excel spreadsheet to be kept in the nursery's records. To do this, they will have to click the "Export" button, and a ".csv" file can be found on the tablet's SD card for them to transfer onto a computer.

ii. Design Decisions

The main decisions that had to be made when developing this application had to do with how the client wanted the usability of the application to be. Through many meetings with my client, the two main things that were always emphasized were ease of use and speed. So, when I began coming up with different design options those two things became my main criteria on judging whether or not a design worked.

The first choice was how I was going to make the database. Having had experience working with SQL databases on an Oracle system, the easiest choice for me to make was to implement my inventory database using the modified version of SQL for Android, called SQLite. The main difference between the regular version of SQL and SQLite was that there is no ability to have a specified primary key in SQLite, and SQLite also makes calls to SQL querying functions to make any modifications to the database. To get around this problem, I just added an automatically incrementing counter that is set to each plant's "ID" value when it is added to the table. This way, you can differentiate each plant from each other because they all have a unique ID number.

The second decision I had to make regarding the design of the application was between barcode and QR code. I found that the easiest way of implementing a barcode/QR code reader was through using the open source XZing project. By implementing their project and using their library functions I was able to successfully implement the scanner. The scanner only has the capability of scanning QR codes, and because this is a mobile application standard type of code to use, the choice of using QR codes over barcodes was made.

The third major design decision I had to make for this application was how to allow for the user to have drop down menu choices, but then also allow for them to add to those options if necessary. My first idea was to just make a selection menu. I thought that this would be the simplest design to implement, and the most efficient when it came to the quickness of use. Unfortunately, if I were to implement drop down menus, it would take away the ability for employees to add things to the options that were specified easily. If menus were implemented it would take them two steps before they reached their outcome (choosing an "other" option, and then typing out their "other" string). This was violating the "ease of use" main criteria for design decisions.

In order to fix this dilemma, after multiple researched ideas and tutorials, I came to the conclusion of using auto complete text fields. What this allowed me to do was implement the values that would be in a drop down menu that Growing Grounds informed me to be the common terms that went into the inventory report, but then it also allowed for me to account for the edge cases where the user would want to type in a value or string that wasn't specified in the drop down menu example. In this case, as soon as a user types in one character from the keyboard (if they follow the guidelines I specify above the text boxes) a drop down list will appear with all of the possible values that Growing Grounds reported to me that were the most commonly used in the current inventory charts. This decision allowed for flexibility, when it came to ease of use; and it also allowed for quickness of execution through the use of the drop down auto complete menus, making it the proper design decision for this application.

The final decision that I had to make pertaining to the design of this project was how to get the information in the generated database from the tablet to the office computers. This decision was easy because of the fact that there is really only one way of solving the problem, which is putting the values into a ".csv" file, or comma separated value file. What this involves is putting in each data entry as a string that is delimited by commas and newlines. When the CSV file is uploaded in Excel, it will take in each value on a new line as a row of the database, where each column is separated by a comma.

When it came to how to transfer this data from the tablet to the nursery's computer, the decision was again very easy because of the limitations of the nursery's equipment. The Growing Grounds nursery does not have internet access, so using email, Drop Box or any other form of media on the internet to save the documents became mute. This lead to saving the collected data in a generated CSV file onto the SD card of the tablet, and then manually taking that file from the tablet to the computer by plugging the tablet into the computer.

In conclusion, the two main determinants that helped me make all of the above design decisions were speed and ease of use. Speed, meaning that the user should never have to take more than thirty seconds at each plant; and ease of use, meaning that anyone of any level of technical knowledge and experience should be able to operate this application without difficulty. The other determinant that forced different design decisions to be made over other more logical ones had to do with the limitations of this application being used onsite at a nursery.

7. System Integration and Testing

The main approach that was taken in the testing of this application was testing in parts. By doing this I was able to ensure that each of the parts of the application worked, and once that was accomplished I was able to confidently move on to integrating all of the parts into a larger system. The goal in testing along the development process was to in essence, have each part tested to the point of satisfactory functionality along the way so that when each part was integrated into the final application it would come together without any issues.

a. Test Plan and Debugging

As mentioned before, the testing of this application was done in parallel to the development of the project. This was broken up into five stages:

i) SQLite Database
ii) QR code reader
iii) Auto-complete text menus
iv) Integrated Application of (i)-(iii)
v) CSV writer

i. SQLite Database

When trying to get a working version of a SQLite database the first step that was taken was to find a tutorial that demonstrated the implementation in such a way that I could make my own version of it. Once that was finished, I made a GrowingGrounds application that consisted of a SQLite database that had a total of four datafields: plantID, text_field_1, text_field_2, text_field3.

The database was built for the plantID field to be an automatically incrementing integer field and the other three text fields to be strings. The database had the ability to type in each of the three fields in an editable text field, add a row to the database, delete a row from the database, edit the row from the database and delete the entire database.

To test the functionality of add, retrieve, update, delete row and delete database, I built the application and ran it. When it was executed for the first time it crashed immediately because I found out, the hard way, that the XML file that you edit (because there is the *Manifest* XML file and an *activity_main* XML file) is very important to the running of your application. When it comes to the graphical layout of your application, although it gives you options for a graphical layout in both possible XML files, you only want to edit the *activity_main* XML file. The *Manifest* XML file has to deal more with permissions pertaining to what your application can and can't access.

The other problem that I had was that I had difficulty printing out and reading in values properly because I wasn't initializing my editable text fields. To make a text field, you have to do 2 things: declare the variable for the field as a global variable within the class, and initialize an instance of the text field that connects it to the text field name that you gave the <EditText> value in your *activity_main* XML file. Once I figured that error out with the text fields, I noticed that the same had to be done for the button fields as well (declaring them as a global variable and then initializing them to the matching named button in the *activity_main* XML file). After fixing those errors (which produced NullPointerException's) the database was able to input data through the add button, retrieve row's values through the retrieve button, re-add an edited entry into the database through the update button, delete a specific row and delete all of the database entries.

ii. QR code reader

The development of this application started the same way as the previous. The difference was that this time I had to dig deeper into the depths of Google to find a good tutorial to aid me in how to incorporate the completed open source XZing project into my project to use the same functionality. A fellow student had posted a tutorial of his own online on how to implement the XZing project in a new application, and I was able to work off of that.

I developed an app to test out the capabilities of the QR code reader. This app had a text field and a button. The text field was set to an instructional string which would be filled with whatever the scan resulted in after the button was pressed. When I built and ran the application it was the first time I had, what I consider, a computer science miracle: the functionality worked perfectly that is until you rotated the application to be in portrait mode.

The app would successfully scan in the value and fill in the text field, but it would only do that when you held the tablet was in landscape mode. The second that you rotated the tablet into portrait mode, the text that was retrieved from the QR code would disappear from the text field and the default instruction string would reappear.

This puzzled me, so I tested it further to see if it was even capable of working in portrait mode. What I found was that if you try and transfer the scanned data into the text field from the QR scanner in portrait mode the application can't do it. After further research I found that QR scanning is meant to only work with the orientation of the device. Because the Galaxy 10.1 tab is oriented in landscape mode, the QR scanning code is only meant to work when the application is in landscape orientation.

To fix this problem I added "android:screenOrientation="landscape"" to the <activity> tag in the *AndroidManifest* XML file. This forced the application to only operate in landscape mode, making it so that the case of rotating the device to lose data would no longer occur.

iii. Auto Complete Text Menus

The same process was taken as mentioned before. I had to find multiple tutorials on StackOverflow and various other blogs and websites that explained the easiest way to implement auto complete menus. After I found a proper tutorial, I made an application with one auto complete editable text field with the functionality built in to list 8 different countries. I built the application and found that the auto complete was a little more complicated that I had originally thought.

The reason is because I had expected that when I typed one character the possible values would come up in a drop down menu, but this didn't happen for the first character, or the second. It wasn't until I typed the third character that a drop down menu would appear. After further examination of the code I saw that you have to set the threshold value for the auto complete text field to be the value of the number of characters that must be typed in order to cause the drop down, with all of the possible values, to pop down. I changed the threshold value to (1) and this gave me the desired results of typing one character and then seeing the possible results that corresponded with that character.

iv. Integrated Application with (i)-(iii)

To integrate the application I first made the graphical layout by building my *activity_main* XML file. I build it with all of the similar entries that I had for the *EditText* fields, *AutoCompleteText* fields, buttons and regular text fields. This worked perfectly, so I moved on to actually putting code into the *MainActivity.java* and *Database.java* files to make the GUI do something.

The first part of code that I added was all of the auto complete arrays for each of the fields. I made each array a global array, and then declared and defined each field within the *MainActivity's oncreate()* method. This proved to be a common error that I continued to make and solve throughout the rest of the integration process, with the error being that any text field or button or thing that can be accessed at anytime CAN NOT be declared inside of the classes functions (like *oncreate()*). These must be made global variables or else they won't remain connected to the text fields throughout the entire lifetime of the application.

The next part that I integrated into the application was the QR code. The problem that I ran into was a similar one that I had when I made the QR application in that I was testing with the tablet in portrait mode and thought that the scanner wasn't working because the text never changed from the instructional default text that I initialized the field to. After remembering what happened in the development of the QR application I made the same change to the *AndroidManifest* XML file and the app was fixed to only operate in landscape mode, eliminating that error.

The last part that I integrated was the SQLite database. The first error that I made was that I put all of the database code in at once instead of gradually adding one function at a time. The problem that this caused is that when I went to press "Add," I thought that it would work because the test application worked, and I essentially just copied it, so when it did nothing I was left stumped. The way that I had to tackle fixing the database was to look and compare line by line the integrated SQLite code to the original SQLite code. Through this comparison I noticed that commas were missing in the "create table" SQL command, and that again I had forgotten to declare all of my buttons as global variables. The other thing I noticed is that because there isn't any real documentation online as to how to upgrade a database to a newer version, if you broke your database you had to actually re-make the project with that new version in order to see if it worked.

The reason for this is because since nobody in the internet universe has bothered to figure out this problem of upgrading databases, if you make big changes (like adding columns or removing columns from the design) the new version of the database cannot overwrite the database that has already been made. It is an unfortunate quirk of SQLite. So after making a new project with the corrected database and all of the other information, I was able to see that the missing commas were extremely significant, and the application worked just as the test application did when they were added.

v. CSV Writer

This was the last feature that I added to the application. I found the openCSV project that was online and downloaded it. I included the JAR library file to the project and then just used a CSVwriter to write to a file that I created on the tablet's SD card. The problem with this is that I don't believe I added the proper JAR file to my project, and I didn't want to add all of the JAR files (when the majority would be useless) in order to find the proper one. As a result I found that to make a ".csv" file, you can just use a FileWriter variable to accomplish the same task as the CSVwriter. The only difference is you just have to specify the extension type on the file when you declare what file you are writing to.

I had to create the file on the SD card because of the fact that the Growing Grounds nursery does not have access to the internet, so options of publishing the files on Drop Box or Google Drive were not available and saving to the disk was my only option. This process worked very easy and I didn't have any problems by following the different posts on StackOverflow on the subject.

b. Analysis and Verification

By choosing a development plan that integrated testing through the process, I felt very confident in the results of my system. I continued to test it out with different QR codes, and saw that it always came up with the proper results, so I knew that my code was valid and that it was a fully functioning application.

Through more testing in an outdoor environment, instead of just inside of my classrooms, I found that there was one problem that would take practice by the user to resolve. The problem when working outdoors is that of the sun. When the sun exposes pictures (like the QR code) to the light that it emits, the tablet's camera has a more difficult time finding the image.

The reason for this is because of the fact that when a computer processes images it looks at the picture as all black and all white. When the sun hits the picture, it "whites it out." This can be seen when you take a picture on your phone and the sun is behind you. In this case, you can no longer see your face, or your face is extremely white because the camera sees the extra light just as white. The way to fix this problem is to make shade over the QR code so that the sun isn't whiting out the image, but also so that there isn't so much shade that the image is being blacked out by the darkness.

After explanation of how to use the system, and a little bit of practice entering in the data fields and navigating through the keyboard, any end user will be able to take down each plants information with quickness and ease.

8. Conclusion

With the knowledge that this project will be further continued for at least another year, I believe that I succeeded in making a very solid base to work off of. I provided the fully functioning database, fully functioning QR code reading and data collection, fully functioning auto complete dropdown menus and a functioning exporter for making the database into a CSV file. The next cycle of the project would only have to add the new ideas that the Growing Grounds representatives come up with, and make it look a little bit more appealing.

The reason that I chose this project was for the community aspect of my college career. Through going to school at Cal Poly and living in San Luis Obispo I have truly appreciated the beautiful community that our school was placed in, and have taken every opportunity given to me to give back to that community. Growing Grounds is an amazing organization that does so much positive work for San Luis Obispo, while also producing beautiful plants and flowers. Through my senior project (The Growing Ground's Inventory Application) life for the employees will hopefully be substantially easier.

Instead of having to go through the entire nursery taking notes on a paper version of the previous weeks inventory list and then spending a couple of hours editing that list on Excel based off of those notes, the process of taking inventory can now be done in one step instead of two. With the addition of my application to the Growing Grounds team, any employee can take inventory with any level of plant knowledge and technical knowledge. By simplifying the data entry process to scanning the name of the plant and then entering in predetermined fields, any employee could perform the inventory task; allowing for the more botanically knowledgeable team members to use their skills towards other projects. The process will also, hopefully, be shortened in the amount of hours it takes to perform the entire inventory taking process. By only having to walk around the nursery and then exporting the database (as opposed to walking around and then Excel data entry), hours could be shaved off of the time it takes to generate the inventory charts, making the nursery more efficient.

As a student I thoroughly enjoyed this project. Developing in the Android environment brought freedom to design and innovate, while also giving me the experience in being self reliant and independent with material that isn't very well documented. But when I look back at this project the two things that I will remember will be the success and the people. The success referring to the project's completion and the people being my wonderful advisor and Growing Grounds clients that I had the pleasure of working with.

So in conclusion, this project will bring the staff of Growing Grounds the ability to grow and become more efficient. By shaving hours off of tasks, such as inventory, employees will be able to get more work done, and there will be less wasted time doing a job twice by the more experienced employees that could be directing their knowledge and skill set to more important projects. This application will be able to bring all of that opportunity to the Growing Grounds staff, and it is very rewarding to know that I was able to help members of my community in bettering a thriving and growing business.

9. Appendices

a. Android Development Environment Set-Up

Step 1: Download the Android ADT bundle. This bundle includes all of the Android specific information that is needed in order to make applications and it also includes the Eclipse IDE, which simplifies the development process. You can find the Android ADT Bundle download here: (http://developer.android.com/sdk/installing/bundle.html).

Step 2: Install the ADT bundle by following the directions that are also found on the previous mentioned webpage.

Step 3: Now you are able to use the Eclipse IDE that is found within this ADT bundle to start developing applications. The next thing that I would strongly encourage would be for you to have an Android device to develop on. If you do not, the emulator tool given by the package does work, but it renders so miserably slow and lacks camera and other capabilities, making it is almost obsolete. If you have a device you just have to plug it into your computer for it to be recognized as usable by the Eclipse IDE for testing.

The one problem that you could potentially run into is installing your devices drivers onto your computer. This can be fixed by visiting the device's manufactures page (Motorola, Samsung etc.) and navigating to the download page. This download page will have different software packages that you can download. You would be interested in the package pertaining to "Device Driver Installation." Once you have downloaded these drivers, plug your device back into the machine you are developing on and it should now recognize your tablet.

The last thing that you have to remember to do when preparing your device for development is to turn on "USB Debugging" in your devices "Settings." When you turn on the USB Debugging it allows for you to upload your project onto your tablet as an actual application and allows for your tablet to communicate to the error reporting tool (logcat) in the Eclipse IDE to better help you debug problems that can occur within your applications.

Step 4: Go on to develop all of your amazing innovative applications!

b. Advice for the Continuation of the Growing Grounds Inventory Application

My number one piece of advice is to find a graphic communications (GRC) major to help you with the design and “pretty” part of your project. Do this AS SOON AS POSSIBLE to ensure that you can find somebody who needs project experience, or needs a senior project themselves so that you aren’t stuck trying to do that all on your own.

The second thing that I would suggest would be to find ways to shield the sun on the tablet. This is because of two problems that the sun could cause; one being that the sun could white out the QR codes, making it impossible to read them, and the second being that the sun on the tablet screen could make it very difficult for the user to see what they are doing on the screen. There is also the problem of weather that I was not able to account for in the fact that if it is raining outside, the staff still has to take inventory, but rain on a tablet is not the best of situations. If you could find a way to fix both of those issues it would improve the user experience dramatically.

c. Selected Source code

Listed in the following pages

MainActivity.Java

```
package com.example.iventorygg;

//These are all of the libraries that had to be imported so that the
//app can function.
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.app.Activity;
import android.view.Menu;

import android.content.Context;
import android.content.Intent;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.WindowManager;
import android.view.inputmethod.InputMethodManager;
import android.widget.Button;
import android.widget.TextView;

import android.util.Log;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;
import android.widget.EditText;

public class MainActivity extends Activity {

        //The following arrays are the possible fields that will come up in the
        //drop down autocomplete menus for Availability, Type, Size, Notes and Price.
        String[] availability =
                {
                        "5",
                        "10",
                        "15",
                        "20",
                        "30",
                        "40",
                        "50+",
                };
        String[] type =
                {
                        " perennial",
                        " native",
                        " succulent",
                        " grass",
                        " edible",
                        " herb",
                        " tree",
                        " fern",
                };
         String[] size =
                 {
                         "1 gal",
                         "2 gal",
                         "3 gal",
                         "5 gal",
                         "15 gal",
                         "4 '' ",
                 };
         String[] notes =
                 {
                         "buds",
                         "bd/blm",
                         "bloom",
                         "fruit",

         };
         String[] price =
                 {
                         "$3.65",
                         "$3.85",
                         "$4.15",
                         "$4.35",
                         "$4.85",
                 };
         // these are the autocomplete text fields for the database add work
         AutoCompleteTextView acTextView1, acTextView2, acTextView3, acTextView4, acTextView5;
         //this is the database object
         Database db;
         //this is the text field that holds the scanned info/name
         TextView tv;
         //this is also used in scanning
         String contents;
         // this is the table that is printed out that shows the database
         TableLayout dataTable;
         //All of the text fields for updating the database
         EditText updateIDField, updateName, updateAvail, updateType, updateSize, updatePrice,
updateNotes;
     //Buttons that are being used
         Button addButt, updateButt, retrieveButt, deleteTable, csvbutt;

         @Override
         protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);

                /*
                 * All of the autocomplete fields for the data entry
                 * The first line declares a new array adapter that matches up to the
                 * previous defined AutoCompleteText views that were globally defined above.
                 *
                 * The second line initializes the text view variable to the graphical autocomplete
                 * text view defined in the activity_main xml file.
                 *
                 * The third line allows for the autocomplete functionality to start after typing in
                 * one character.
                 *
                 * The fourth line says that the autocomplete text field is connected to have the dropdown
menu
                 * that you made in the first line.
                 */
                ArrayAdapter<String> adapter1 = new
ArrayAdapter<String>(this,android.R.layout.simple_dropdown_item_1line,availability);
         acTextView1 = (AutoCompleteTextView)findViewById(R.id.Availability);
         acTextView1.setThreshold(1);
         acTextView1.setAdapter(adapter1);

                ArrayAdapter<String> adapter2 = new
ArrayAdapter<String>(this,android.R.layout.simple_dropdown_item_1line,type);
         acTextView2 = (AutoCompleteTextView)findViewById(R.id.Type);
         acTextView2.setThreshold(1);
         acTextView2.setAdapter(adapter2);

                ArrayAdapter<String> adapter3 = new
ArrayAdapter<String>(this,android.R.layout.simple_dropdown_item_1line,size);
         acTextView3 = (AutoCompleteTextView)findViewById(R.id.Size);
         acTextView3.setThreshold(1);
         acTextView3.setAdapter(adapter3);
```

```
		ArrayAdapter<String> adapter4 = new
ArrayAdapter<String>(this,android.R.layout.simple_dropdown_item_1line,notes);
	acTextView4 = (AutoCompleteTextView)findViewById(R.id.Notes);
	acTextView4.setThreshold(1);
	acTextView4.setAdapter(adapter4);

		ArrayAdapter<String> adapter5 = new
ArrayAdapter<String>(this,android.R.layout.simple_dropdown_item_1line,price);
	acTextView5 = (AutoCompleteTextView)findViewById(R.id.Price);
	acTextView5.setThreshold(1 );
	acTextView5.setAdapter(adapter5);

	/* the following lines define the EditText fields to match up with their corresponding
	 * actual objects that were declared in the activity_main.xml file.
	 */
	updateIDField=			(EditText)findViewById(R.id.update_id_field);
	updateName=		(EditText)findViewById(R.id.uName);
	updateAvail=		(EditText)findViewById(R.id.uAvail);
	updateType=		(EditText)findViewById(R.id.uType);
	updateSize=			(EditText)findViewById(R.id.uSize);
	updateNotes=		(EditText)findViewById(R.id.uNotes);
	updatePrice=		(EditText)findViewById(R.id.uPrice);

	// Initializing the scanning button to be scannable
	Button btn = (Button) findViewById(R.id.scanButton);
	// Making a listener for when the button is pressed
	btn.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				//open the activity when clicked
				Intent intent = new Intent("com.google.zxing.client.android.SCAN");
				intent.putExtra("SCAN_MODE", "QR_CODE_MODE");
				startActivityForResult(intent, 0);
			}
	});

	/* Initializes all of the other buttons in the GUI that
	 * was declared in the activity_main.xml file
	 */
	addButt = (Button) findViewById(R.id.add);
	retrieveButt = (Button) findViewById(R.id.retrieve_button);
	updateButt = (Button) findViewById(R.id.update_button);
	deleteTable = (Button) findViewById(R.id.delete_table_button);
	csvbutt = (Button) findViewById(R.id.csv_button);

	/* The folowing setOnClickListener statemenets are made to
	 * make it so that when each button is pressed the functions within
	 * this statement will be called to do work.
	 */
	addButt.setOnClickListener
	(
		new View.OnClickListener()
		{
				@Override public void onClick(View v) {addRow();}
			}
	);
	updateButt.setOnClickListener
	(
	new View.OnClickListener()
		{
		@Override public void onClick(View v) {updateRow();}
		}
	);

	retrieveButt.setOnClickListener
	(
	new View.OnClickListener()
	{
				@Override public void onClick(View v) {retrieveRow();}
```

```
                    }
);
deleteTable.setOnClickListener
(
new View.OnClickListener()
{
                              @Override public void onClick(View v) {deleteTable();}
                    }
 );
csvbutt.setOnClickListener
(
    new View.OnClickListener()
    {
                              @Override public void onClick(View v) {makecsv();}
    }
 );

// THE DATA TABLE
dataTable= (TableLayout)findViewById(R.id.data_table);
db = new Database(this);
// load the databases table to the screen
updateTable();
}

/* This function adds a row of data to the Plant Inventory Database*/
private void addRow()
{
        try
{
        // ask the database manager to add a row given the two strings
        db.addRow
        (
                        tv.getText().toString(),
                        acTextView1.getText().toString(),
                        acTextView2.getText().toString(),
                        acTextView3.getText().toString(),
                        acTextView4.getText().toString(),
                        acTextView5.getText().toString()
        );

        updateTable();
        // remove all user input from the Activity
        emptyFormFields();
}
catch (Exception e)
{
        Log.e("Add Error", e.toString());
        e.printStackTrace();
}

}

/* This function allows the user to pull up an entry and edit it to be
 * re-inserted into the table.
 */
private void updateRow()
{
try
{
        // This puts the updated/edited text to make up the new edited
        //row back into the database.
        db.updateRow
        (
                Long.parseLong(updateIDField.getText().toString()),
                updateName.getText().toString(),
                updateAvail.getText().toString(),
                updateType.getText().toString(),
                updateSize.getText().toString(),
                updateNotes.getText().toString(),
                    updatePrice.getText().toString()
            );

            // request the table be updated
                    updateTable();

                    // remove all user input from the Activity
            emptyFormFields();
    }
    catch (Exception e)
    {
            Log.e("Update Error", e.toString());
            e.printStackTrace();
    }
    }

    /* This function gets the specified row from the database and then puts that
     * information into the update text fields.
     */
    private void retrieveRow()
    {
    try
    {
            // The ArrayList that holds the row data
            ArrayList<Object> row;
            // Get the row wiht the given rowID from the database.
            row = db.getRowAsArray(Long.parseLong(updateIDField.getText().toString()));

            // fill in the update text fields with the retrieved data
            updateName.setText((String)row.get(1));
            updateAvail.setText((String)row.get(2));
            updateType.setText((String)row.get(3));
            updateSize.setText((String)row.get(4));
            updateNotes.setText((String)row.get(5));
            updatePrice.setText((String)row.get(6));
    }
    catch (Exception e)
    {
            Log.e("Retrieve Error", e.toString());
            e.printStackTrace();
    }
    }

    /* This function will delete all of the rows in the database */
    private void deleteTable()
    {
   db.deleteDB();
   updateTable();
    }

    /* This funciton will update the table that is being displayed on the bottom of the
     * applications screen.
     */
private void updateTable()
{
    // this was a little detail I found in a tutorial, doesn't make a lot of sense,
    //but is in place for clean up purposes.
    while (dataTable.getChildCount() > 1)
    {
            // while there are at least two rows in the table widget, delete
            // the second row.
            dataTable.removeViewAt(1);
    }

    // get all of the rows from the database.
    ArrayList<ArrayList<Object>> data = db.getAllRowsAsArrays();

    // go through data and print out each of the fields into the proper
    //column in the table.
    for (int position=0; position < data.size(); position++)
    {
            // makes a table row to put the information.
            TableRow tableRow= new TableRow(this);

        //gets the first row from the database.
            ArrayList<Object> row = data.get(position);

        //makes a text view to put the information for ID
            //to add to the table. The following blocks do the same for all of the
            //other columns values in the row
            TextView idText = new TextView(this);
            idText.setText(row.get(0).toString());
            tableRow.addView(idText);

            TextView nameText = new TextView(this);
            nameText.setText(row.get(1).toString());
            tableRow.addView(nameText);

            TextView availText = new TextView(this);
            availText.setText(row.get(2).toString());
            tableRow.addView(availText);

            TextView typeText = new TextView(this);
            typeText.setText(row.get(3).toString());
            tableRow.addView(typeText);

            TextView sizeText = new TextView(this);
            sizeText.setText(row.get(4).toString());
            tableRow.addView(sizeText);

            TextView notesText = new TextView(this);
            notesText.setText(row.get(5).toString());
            tableRow.addView(notesText);

            TextView priceText = new TextView(this);
            priceText.setText(row.get(6).toString());
            tableRow.addView(priceText);

            // this will add the text view to the table.
            dataTable.addView(tableRow);
    }
}

/* this function will clear the datafields of text after the button has been
 * used and the information is put into the database.
 */
private void emptyFormFields()
{
    tv.setText("Click to Scan Name -->");
    acTextView1.setText("");
    acTextView2.setText("");
    acTextView3.setText("");
    acTextView4.setText("");
    acTextView5.setText("");

}

/* This funciton generates a .csv file with all of the database rows in it */
private void makecsv()
{
    try{
       /* makes a file writer that points to the storage on the
        * sd card to make a new file called Inventory.csv
        */
             FileWriter filewrite = new FileWriter("/sdcard/Inventory.csv");
             ArrayList<ArrayList<Object>> data = db.getAllRowsAsArrays();

             // this is the Header of all the Column Labels for the Database.
```

```
                    filewrite.append("Plant Name (Botanical | Common)");
                filewrite.append(',');
                filewrite.append("Availability");
                filewrite.append(",");
                filewrite.append("Type");
                filewrite.append(",");
                filewrite.append("Size");
                filewrite.append(",");
                filewrite.append("Notes");
                filewrite.append(",");
                filewrite.append("Price");
                filewrite.append('\n');

                // This iterates through each row in the database and prints out
                //all of the information in the same format as above.
                   for (int position=0; position < data.size(); position++)
                   {
                          ArrayList<Object> row = data.get(position);
                            filewrite.append(row.get(1).toString());
                        filewrite.append(',');
                        filewrite.append(row.get(2).toString());
                        filewrite.append(",");
                        filewrite.append(row.get(3).toString());
                        filewrite.append(",");
                        filewrite.append(row.get(4).toString());
                        filewrite.append(",");
                        filewrite.append(row.get(5).toString());
                        filewrite.append(",");
                        filewrite.append(row.get(6).toString());
                        filewrite.append('\n');
                   }
                    //the file writer is flushed and then closed since you are done writing.
                filewrite.flush();
                filewrite.close();
        }
        catch(IOException e){
                e.printStackTrace();
        }

    }

    // This is the result for pressing the "Scan Button"
    // Calls the intent of the QR scanner to retrieve the data
    //fromt he qr code for use.
    public void onActivityResult(int requestCode, int resultCode, Intent intent) {
                if (requestCode == 0) {
                        if (resultCode == RESULT_OK) {
                                contents = intent.getStringExtra("SCAN_RESULT");

                                //set the text view to the results
                                tv = (TextView) findViewById(R.id.scan);
                                tv.setText(contents);
                        } else if (resultCode == RESULT_CANCELED) {
                                // No image found, back button pressed
                        }
                }
        }

    // This funciton makes it so that the keyboard will only pop up when the text fields are
    //selected and it will be hidden once you click off of the text views.
    private boolean isTouchInsideView(final MotionEvent ev, final View currentFocus) {
        final int[] loc = new int[2];
        currentFocus.getLocationOnScreen(loc);
        return ev.getRawX() > loc[0] && ev.getRawY() > loc[1] && ev.getRawX() < (loc[0] +
currentFocus.getWidth())
            && ev.getRawY() < (loc[1] + currentFocus.getHeight());
    }

    //This overrides the touch even that is defaulted to keep the keyboard up when a text field is being
    //used.
        @Override
        public boolean dispatchTouchEvent(final MotionEvent ev) {
            // all touch events close the keyboard before they are processed except EditText instances.
            // if focus is an EditText we need to check, if the touchevent was inside the focus editTexts
            final View currentFocus = getCurrentFocus();
            if (!(currentFocus instanceof EditText) || !isTouchInsideView(ev, currentFocus)) {
                ((InputMethodManager)
getApplicationContext().getSystemService(Context.INPUT_METHOD_SERVICE))
                    .hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
InputMethodManager.HIDE_NOT_ALWAYS);
            }
            return super.dispatchTouchEvent(ev);
        }
}
```

Database.java

```
package com.example.iventorygg;

import java.util.ArrayList;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

// this is the Database that does all of the work for the Inventory application.
public class Database {
        // the Activity or Application that is creating an object from this class.
                Context context;

                // a reference to the database used by this application/object
                private SQLiteDatabase db;

                // These constants are specific to the database.  They should be
                // changed to suit your needs.
                private final String DB_NAME = "database_name";
                private final int DB_VERSION = 1;

                // These constants are specific to the database table.  They should be
                // changed to suit your needs.
                private final String TABLE_NAME = "database_table";
                private final String TABLE_ROW_ID = "id";
                /* latin plant name */
                private final String TABLE_PNAME = "table_pname";
                /* avaiable plant # */
                private final String TABLE_AVAIL = "table_avail";
                /*type */
                private final String TABLE_TYPE = "table_type";
                /* size */
                private final String TABLE_SIZE = "table_size";
                /*notes*/
                private final String TABLE_NOTES = "table_notes";
                /*price*/
                private final String TABLE_PRICE = "table_price";

                //This funciton creates an object of type Database.
                public Database(Context context)
                {
                        this.context = context;

                        // create or open the database
                        CustomSQLiteOpenHelper helper = new CustomSQLiteOpenHelper(context);
                        this.db = helper.getWritableDatabase();
                }

                /* ADDING A ROW TO THE DATABASE TABLE */
                public void addRow(String row1, String row2, String row3, String row4, String row5, String
row6)
                {
                        // this is a key value pair holder used by android's SQLite functions
                        ContentValues values = new ContentValues();
                        //This puts the string values into their corresponding database columns.
                        values.put(TABLE_PNAME, row1);
                        values.put(TABLE_AVAIL, row2);
                        values.put(TABLE_TYPE, row3);
                        values.put(TABLE_SIZE, row4);
                        values.put(TABLE_NOTES, row5);
                        values.put(TABLE_PRICE, row6);
```

```
		// ask the database object to insert the new data
			try{db.insert(TABLE_NAME, null, values);}
			catch(Exception e)
			{
				Log.e("DB ERROR", e.toString());
				e.printStackTrace();
			}
		}

		/* UPDATES A ROW IN THE DATABASE */
		public void updateRow(long rowID, String row1, String row2, String row3, String row4,
						String row5, String row6)
		{
			// this is a key value pair holder used by android's SQLite functions
			ContentValues values = new ContentValues();
			//this puts the given strings into their corresponding database columns.
			values.put(TABLE_PNAME, row1);
			values.put(TABLE_AVAIL, row2);
			values.put(TABLE_TYPE, row3);
			values.put(TABLE_SIZE, row4);
			values.put(TABLE_NOTES, row5);
			values.put(TABLE_PRICE, row6);

			// ask the database object to update the database row of given rowID
			try {db.update(TABLE_NAME, values, TABLE_ROW_ID + "=" + rowID, null);}
			catch (Exception e)
			{
				Log.e("DB Error", e.toString());
				e.printStackTrace();
			}
		}

		// This funciton will clear all of the rows in the database.
		public void deleteDB()
		{
			db.delete(TABLE_NAME, null, null);
		}

		/*RETRIEVING A ROW FROM THE DATABASE TABLE*/
		public ArrayList<Object> getRowAsArray(long rowID)
		{
			// create an array list to store data from the database row.
			ArrayList<Object> rowArray = new ArrayList<Object>();
			Cursor cursor;

			try
			{
				// this is a cursor used to get the desired row from the database.
				cursor = db.query
				(
						TABLE_NAME,
						new String[] { TABLE_ROW_ID, TABLE_PNAME, TABLE_AVAIL,
TABLE_TYPE, TABLE_SIZE,
								TABLE_NOTES, TABLE_PRICE},
						TABLE_ROW_ID + "=" + rowID,
						null,
						null,
						null,
						null,
						null
				);

				// move the pointer to position zero in the cursor.
				cursor.moveToFirst();

				// if there is data available after the cursor's pointer, add
				// it to the ArrayList that will be returned by the method.
				if (!cursor.isAfterLast())
				{
```

```
                                do
                                {
                                        rowArray.add(cursor.getLong(0));
                                        rowArray.add(cursor.getString(1));
                                        rowArray.add(cursor.getString(2));
                                        rowArray.add(cursor.getString(3));
                                        rowArray.add(cursor.getString(4));
                                        rowArray.add(cursor.getString(5));
                                        rowArray.add(cursor.getString(6));

                                }
                                while (cursor.moveToNext());
                        }

                        // done using the cursor
                        cursor.close();
                }
                catch (SQLException e)
                {
                        Log.e("DB ERROR", e.toString());
                        e.printStackTrace();
                }

                // return the ArrayList containing the given row from the database.
                return rowArray;
        }



        /*RETRIEVING ALL ROWS FROM THE DATABASE TABLE */
        public ArrayList<ArrayList<Object>> getAllRowsAsArrays()
        {
                // create an ArrayList that will hold all of the data collected from
                // the database.
                ArrayList<ArrayList<Object>> dataArrays = new ArrayList<ArrayList<Object>>();

                // this is a cursor used to iterate through the database.
                Cursor cursor;

                try
                {
                        // ask the database object to create the cursor.
                        cursor = db.query(
                                        TABLE_NAME,
                                        new String[]{ TABLE_ROW_ID, TABLE_PNAME, TABLE_AVAIL,
TABLE_TYPE,
                                                                TABLE_SIZE, TABLE_NOTES,
TABLE_PRICE},
                                        null,
                                        null,
                                        null,
                                        null,
                                        null,
                                        null
                        );

                        // move the cursor's pointer to position zero.
                        cursor.moveToFirst();

                        // if there is data after the current cursor position, add it
                        // to the ArrayList.
                        if (!cursor.isAfterLast())
                        {
                                do
                                {
                                        ArrayList<Object> dataList = new ArrayList<Object>();

                                        dataList.add(cursor.getLong(0));
                                        dataList.add(cursor.getString(1));
                                        dataList.add(cursor.getString(2));
                                        dataList.add(cursor.getString(3));
                                        dataList.add(cursor.getString(4));
                                        dataList.add(cursor.getString(5));
                                        dataList.add(cursor.getString(6));

                                        dataArrays.add(dataList);
                                }
                                // move the cursor's pointer up one position.
                                while (cursor.moveToNext());
                        }
                }
                catch (SQLException e)
                {
                        Log.e("DB Error", e.toString());
                        e.printStackTrace();
                }

                // return the ArrayList that holds the data collected from
                // the database.
                return dataArrays;
        }

        /* Defines the actual database with helper calls to actual SQL funcitons */
        private class CustomSQLiteOpenHelper extends SQLiteOpenHelper
        {
                public CustomSQLiteOpenHelper(Context context)
                {
                        super(context, DB_NAME, null, DB_VERSION);
                }

                @Override
                public void onCreate(SQLiteDatabase db)
                {
                        // This string is used to create the database.  It should
                        // be changed to suit your needs.
                        String newTableQueryString = "create table " +
                                TABLE_NAME +
                                " (" +
                                TABLE_ROW_ID + " integer primary key autoincrement
                                                             not null," +
                                TABLE_PNAME + " text," +
                                TABLE_AVAIL + " text," +
                                TABLE_TYPE + " text," +
                                TABLE_SIZE + " text," +
                                TABLE_NOTES + " text," +
                                TABLE_PRICE + " text" +
                                ");";
                        // execute the query string to the database.
                        db.execSQL(newTableQueryString);
                }


                @Override
                public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
                {
                        // NOTHING TO DO HERE. THIS IS THE ORIGINAL DATABASE VERSION.
                        // OTHERWISE, YOU WOULD SPECIFIY HOW TO UPGRADE THE DATABASE.
                }
        }
        }
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <LinearLayout
            android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
         <TextView
            android:id="@+id/scan"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click to Scan Names -->" />
         <TextView
                android:text=" "
                android:layout_width="60dp"
                android:layout_height="wrap_content"
                />

        <Button
        android:id="@+id/scanButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_alignParentTop="true"
        android:text="Scan" />
    </LinearLayout>
    <TextView
                android:text=" "
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
        />
    <TextView
                android:text="# Availability (5-50+) ---- Type(space bar for options) ------- Size (1-15
gal) ----------- Notes (bud/bloom/fruit) ----------- Price"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
        />
        <LinearLayout
            android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >

                <AutoCompleteTextView
                    android:id="@+id/Availability"
                    android:layout_width="100dp"
                    android:layout_height="wrap_content" >
                </AutoCompleteTextView>
                <TextView
                android:text=" "
                android:layout_width="70dp"
                android:layout_height="wrap_content"
                />
                <AutoCompleteTextView
                    android:id="@+id/Type"
                    android:layout_width="160dp"
                    android:layout_height="wrap_content" >
                </AutoCompleteTextView>
                <TextView
                android:text=" "
                android:layout_width="40dp"
                android:layout_height="wrap_content"
                />
                <AutoCompleteTextView
                android:id="@+id/Size"
                android:layout_width="100dp"
                android:layout_height="wrap_content" >
            </AutoCompleteTextView>
            <TextView
            android:text="  "
            android:layout_width="60dp"
            android:layout_height="wrap_content"
            />
            <AutoCompleteTextView
                android:id="@+id/Notes"
                android:layout_width="140dp"
                android:layout_height="wrap_content" >
            </AutoCompleteTextView>
            <TextView
            android:text="  "
            android:layout_width="60dp"
            android:layout_height="wrap_content"
            />
            <AutoCompleteTextView
                android:id="@+id/Price"
                android:layout_width="100dp"
                android:layout_height="wrap_content" >
            </AutoCompleteTextView>
            <Button
        android:id="@+id/add"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Add" />
    </LinearLayout>
    <TextView
            android:text="  "
            android:layout_width="50dp"
            android:layout_height="wrap_content"
            />
<TextView
            android:text="Pick id to edit ------------------------- Name ------------------------------
-------------------- Availability--------- Type ------------- Size ------------- Notes -------------
Price"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
    />
    <LinearLayout
            android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
    <EditText
            android:id="@+id/update_id_field"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:minWidth="45px"
                    />
             <Button
            android:id="@+id/retrieve_button"
            android:text="Retrieve"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            />

            <EditText
            android:id="@+id/uName"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:minWidth="350px"
                    />
       <EditText
            android:id="@+id/uAvail"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
                android:minWidth="100px"
                        />
            <EditText
                android:id="@+id/uType"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:minWidth="115px"
                        />
            <EditText
                android:id="@+id/uSize"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:minWidth="100px"
                        />
            <EditText
                android:id="@+id/uNotes"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:minWidth="100px"
                        />
            <EditText
                android:id="@+id/uPrice"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:minWidth="100px"
                        />
            <Button
                android:id="@+id/update_button"
                android:text="Update"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                />
                 </LinearLayout>


    <Button
                android:id="@+id/delete_table_button"
                android:text="                                   Delete Database Table
"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                />
    <Button
                android:id="@+id/csv_button"
                android:text="                                   Make Excel File
"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                />
    <TextView
        android:text="  "
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        />
        <TextView
        android:text="------------------------------------------------------------------------Plant
Availabilty Table ---------------------------------------------------------------------------------
-"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        />

        <!-- THE DATA TABLE -->
        <TableLayout
                android:id="@+id/data_table"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                >
                <TableRow>
```

```
                        <TextView
                                android:text="ID"
                                android:minWidth="50px"
                                />
                        <TextView
                                android:text="Plant Name"
                                android:minWidth="225px"
                                />
                        <TextView
                                android:text="Availability"
                                android:minWidth="125px"
                                />
                        <TextView
                                android:text="Type"
                                android:minWidth="125px"
                                />
                        <TextView
                                android:text="Size"
                                android:minWidth="125px"
                                />
                        <TextView
                                android:text="Notes"
                                android:minWidth="125px"
                                />
                        <TextView
                                android:text="Price"
                                android:minWidth="125px"
                                />
                </TableRow>
        </TableLayout>




</LinearLayout>
```

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.iventorygg"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="16" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:screenOrientation="landscape"
            android:name="com.example.iventorygg.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```