

# **Tracket Inventory Application for Tennis**

Jeremy Sagaille  
Professor Traver  
ECE 499  
March 17, 2015

## Table of Contents

Preface.....	3
Introduction.....	4
Background.....	6
Design Specifications.....	10
Design Alternatives.....	15
Preliminary Proposed Design.....	20
Final Design and Implementation.....	26
Performance Estimates and Results.....	37
Production Schedule.....	40
Cost Analysis.....	42
User’s Manual.....	43
Discussion, Conclusions, and Recommendations.....	45
References.....	48
Appendices.....	50

## Figures and Tables

Figure 1.....	12
Figure 2.....	16
Figure 3.....	16
Figure 4.....	17
Figure 5.....	20
Figure 6.....	21
Figure 7.....	21
Figure 8.....	22
Figure 9.....	23
Figure 10.....	24
Figure 11.....	27
Figure 12.....	28
Figure 13.....	29
Figure 14.....	30
Figure 15.....	31
Figure 16.....	32
Figure 17.....	33
Figure 18.....	34
Figure 19.....	35
Figure 20.....	38

## Preface

The students of Union College are immersed in quite a diverse array of majors and interests. During my four years here I have learned a lot about my self and my own interests as well as a lot about my fellow classmates and their interests. I would say that one of the most significant things that I have learned and noticed repeatedly is that no one is an expert in everything. When we need to get something done especially in engineering, we rely on past experiences and problem solving skills to complete the task. Often times, the problem we are trying to solve exceeds the boundaries of our current knowledge base, therefore, we reach out to friends and colleagues with more experience in the subject to help expedite the process and ensure the problem is solved in the most optimum way possible.

Seeing professors and other professionals use this tactic to complete tasks has been the inspiration for my senior project. My younger brother, Joshua, is an avid tennis player and entrepreneur. He uses the connections he's made to help subsidize the cost of training and competing. Currently, his largest source of income is stringing the rackets of the people he knows and the people he plays with. He is quite efficient with stringing and providing his various racket services and I think if he were to keep better tabs on pricing, overhead, and materials being purchased and consumed, he could further increase his profit margin, which is always the goal of any successful business.

## Introduction

The overall goal of this project is to increase the profit margin of my brothers business. Joshua currently has no method of tracking his materials, sales, or profits and occasionally, he strings rackets for people free of charge if the strings break within a certain time period. Currently, his customer base is small enough that he generally remembers the time frame of when he strings rackets but as his customer demographic grows he may have a more trouble keeping track, which could shrink profit his margin.

The idea for the project is a Mobile RFID Inventory System specifically for tennis rackets and the racket technicians that are tasked with stringing and servicing a large amount of rackets. The idea is that a racket technician would use the system to keep an electronic trail of what jobs they are performing for which customers and be able to track the materials that they use, what they charge, and any other pertinent information that the technician would like to keep track of. The project will consist of two main components, the hardware (NFC Tags) and the software (the Android application).

The traditional way of tracking the data of a previous string job was by putting a simple sticker on the racket with the most current string job info written on it. The idea is to digitize this process and make it better. Next adding an NFC tag to the rackets

that would store a serial number in which the technician would then be able to read from an NFC and Android enabled application that would catalogue the certain services performed. Designing appropriate and effective software will be the bulk of this project since the hardware portion for the most part already exists and is very cost effective.

## Background

For the project a Mobile NFC Inventory System specifically for tennis rackets and the racket stringing technicians will be made. The system will be capable of being tasked with servicing a large amount of rackets. The idea is that a racket technician would use the system to keep a digital paper trail of what jobs they are performing for each customer and be able to track the materials that they use, their charge, and other pertinent information that the technician would like to keep track of. While this is quite a unique idea, there are still numerous sources available on the individual sub-problems that this project is made up of. The resources found may prove very helpful in the near future.

The first thing researched was current inventory systems, how they work, and why we use them. What the research turned up was very interesting and may prove to be profitable to the customer the product is being designed for. The whole reason we go into business is to make a profit, and in order for businesses to be as successful as possible, they need to be as efficient as possible. The best way to increase efficiency is to decrease waste by stripping away unnecessary expenditures or by redirecting improperly delegated resources [1]. Essentially, by implementing inventory systems we can look at raw data and see where we can improve aspects of the business model, such as, by looking at the most efficient way of buying supplies. In the case of the tennis racket system, we could look to see what we can find in bulk quantities and

when we should purchase more of these items in order to get the best price [1]. Analyzing and using the data that is provided from inventory systems generally proves useful in increasing efficiency and optimizing the growth of profits. Although, inventory systems do have many pros, there are some cons that are worth noting pertaining to RFID systems that an engineer should take into account while trying to improve on previous models.

The use of RFID inventory systems in some cases has improved upon how traditional line-of-sight systems work such as barcode or QR code systems. Eliminating line-of-sight increases productivity because it takes less time and accuracy to interact with marked items. Where as with a barcode, scanning it is necessary but sometimes it doesn't scan and you have trouble lining it up making the whole process a huge hassle. There are also no orientation limitations with RFID, which can speed up the identification process. Furthermore, RFID's ability to scan multiple tags simultaneously increases speed and productivity [2]. Another upside of RFID systems is the security that is built in. Since, RFID tags are virtually impossible to replicate or impersonate, it is more secure because barcodes can be copied and replicated with relative ease. The largest, and really main, obstacle of RFID is the price. Traditional RFID tags cost anywhere from 15-75 cents a piece and the hardware needed to interface with the tags that renders the systems useful also can be pricey. Obviously the prices depend on the quantity being purchased, but even with that fact, the overhead of line of sight systems is far cheaper then that of RFID [2]. Barcode systems only need scanners and then you can print personal barcodes out. With respect to the

project that will be worked on, a traditional line-of-sight system would definitely be cheaper and easier to cross platforms but they aren't as cool and take away from the presentation aspect if the racket has an ugly barcode showing. Having a tag on the exterior of a racket also exposes it to the wear and tear of usage, and may render the barcode unreadable.

Next, the reason to create a mobile inventory system is because it is convenient and because nowadays, most people own a smart phone, which decreases direct costs of the system [3]. Since the devices are personal the systems will also be more secure. Now that all of the hardware has been described and established as safe, secure, and relatively cost effective, next onto hardware control. The device family that will be used is the Samsung Galaxy. These devices have NFC (Near Field Communication), a form of RFID communication, built in. Since these devices run Android, the software that will need to be written to get this project to work will need to be in the form of an android application. While looking for background information on Android and NFC applications it was discovered that an entire textbook in PDF form was online dedicated to the subject. Surely this book will aid greatly in the coming months considering the difficulty in creating Android Apps and interfacing them with NFC. This book explicitly takes you through the history of NFC and all of the minute technical details. It then talks about app development and most importantly how to combine the two technologies [4]. This book will prove to be extremely helpful as it contains a lot of information on Android and NFC unlike most of the documents that were discovered.



Finally, this entire project will come together because of the base technology. NFC and RFID are two very popular technologies that are up and coming and most likely here for the long run. In the future, more and more things in everyday life will become automated. As technology progresses, the result will be a decrease in human control and interaction, devices will get “smarter” thus, less human time will be spent, making us more productive as a society [5]. At the forefront of this push are technologies such as RFID and NFC. These types of technologies contribute toward our never-ending search of ways to make our lives easier. If implemented correctly, the NFC technology paired with appropriate software will make my project a success and this project change the way tennis racket technicians do business.

## Design Specifications

After careful consideration of what is actually practical for the scope of this project the following list is what was put together and it is what will be achieved when the project is over.

- The implanted NFC Tag must be undetectable by the tennis player.
- The device must easily read the tag's serial number.
- The application must be easy to use.
- The application should be able to create a new tag.
- If a tag is somehow lost or damaged then the old serial number can be written on to a new tag.
- The app must track sales and profits.
- The app must track materials used.
- The app must track customer preferences on racquet jobs along with a history of jobs done.
- Include a way of backing up the data.
- Secure the data.

Using the bullet list above as guidance each point can be gone through one by one to go deeper into the design and implementation of each step. There are two main sections to this project the hardware side and then the larger software portion.

Starting with the hardware portion and our first requirement, the tag must be undetectable to the tennis player that is using the racket. In this regard there is not much work to be done. The world of NFC has built off of RFID and the technology that is currently available to us actually exceeds the needs of the project. The average bare bones NFC tag on the market today weighs less than 6 grams, less than 1% of the average weight of a strung racket. Not only is the weight essentially negligible, another aspect is that the tags will be placed in the handle of the racket making the tag virtually unnoticeable to the player, accomplishing the initial goal.

The next requirement is that, once placed, the tags will be easily scanned with close to 100% reliability and accuracy. A factor that could encourage unreliable communication is electromagnetic interference (EMI). However, no materials in the handles of tennis rackets are known to cause EMI since most handles are constructed from plastic and fiberglass [6]. Along with research and rigorous testing the conclusion is that the handle of a tennis racket contains no elements that would cause EMI. This would make our goal of accurately and easily scanning the tag completely plausible.

The next portions of the design requirements pertain to the project's software requirements. The role of software is to simplify and to save us time in the long run. If the software we are using is not effective in saving us time and money, there is no reason to use it. My overall goal of the application is to be secure, effective, easy to use, and to track the appropriate data. My goal for the application is to create the simplest yet most dependable piece of software possible. Goal number one is making sure that the software is easy to use. If the software is not easy to use people will not use it, plain and simple.

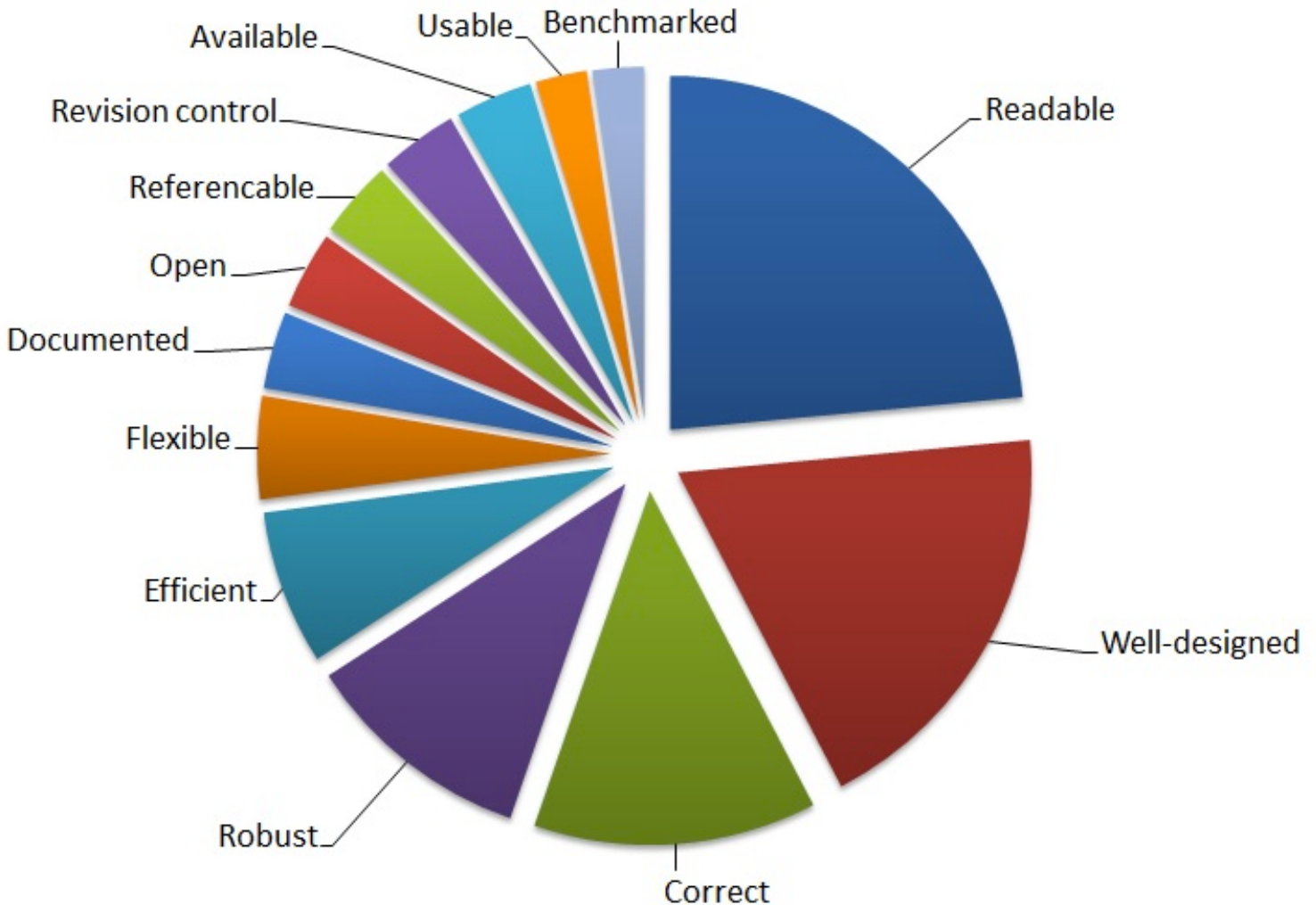


Figure 1. What makes good code [7]

Creating good software is no easy task. By looking at Figure 1 above we can see that good software is made up of a lot of pieces. The plan is to implement as many pieces of this figure as possible starting from largest to smallest.

The first portions that are most essential for the success of the project are aesthetics, design, and correctness of the program. As far as aesthetics, the first and most important part of a successful enterprise is branding. Since graphic design is not my expertise, outside help was brought in, in the form of a classmate of mine with a visual arts background to help with this task. Together we will do our best to come up with a design that will not only get the app idea across but be unique and exciting at the same time.

After developing the logo, the next most important piece is the splash screen; known as the first screen you see when you open the program. In the case of this application, a login screen, a main screen, and then a home screen for more security was chosen since the application will contain sensitive information. To keep the application streamlined and simple the home screen is only to have three options to start with. The three most important things that the application must do are going to be the options on the home screen. The first option, and probably the most important one will be scanning an existing tag. The screen that this button links to will allow the

racket technician to scan the racket and open up the owners file. The second most important is being able to view and search through client database, therefore, the second option will be a database option. The screen that will be linked to this option will open up a list view of all the clients in the database, along with a search bar and an option to add a new customer at the top. Finally, the main reason that the application is being created is to track customer data and trends. The final button on the home screen will be the display data option. The screen that will be linked to this button will contain all the stats that can be calculated using the data inputted by the racket technician. The data that will be displayed will be a range of facts from economic to pure totals. Numbers like gross and net profit, average profit per string job, number of jobs completed, yards of string used, total overhead, graphs of when the busiest times of the year are, etc.. My goal is to make the data open to the racket tech so they can include whatever data they would like to and it can be tracked and used to improve business and productivity.

Finally, the requirements are meant to be a set of guidelines. As the project evolves adding features to the application as are necessary is the practice that will keep the software current. The requirements and goals are laid out to get me to certain points so that the scope of my software can be broadened and hopefully reach as many customers as possible.

## Design Alternatives

In this section we will discuss alternatives to certain choices we made in the design specifications section. The discussion will also include why choices from the previous section are the best possible choices by showing what the alternatives lack. Starting off with the same order of hardware then software, we will first discuss why the hardware decisions chosen were the right choices.

Beginning with the identification choice, there are quite a few ways of identifying things. However, to narrow things down, creating a mobile digital identification system was the idea in mind. The three main digital identification mediums that are currently available are; Quick Response Code or QR codes, Radio Frequency Identification or RFID, and finally a subset of RFID called Near Field Communication or NFC. Right off of the bat we can throw out RFID. Full RFID tags tend to be bigger, bulkier, and require more power, which is not ideal for the application at hand. Since we are down to QR and NFC let's talk about QR is first before comparing the two. "QR codes are two-dimensional bar codes that can contain any alphanumeric text and often feature URLs that direct users to sites in which they can learn about an object or place (a practice known as "mobile tagging")" [10]. The QR code is very similar to the traditional barcode, requires line of site, and requires an appropriate scanner in order to decode the data. That being said let's look at a couple of figures to show us what NFC and QR look like when they go head to head. Please refer to the following two figures 2, and 3.

## Tracket Inventory System



Figure 2. NFC vs Barcode [9]

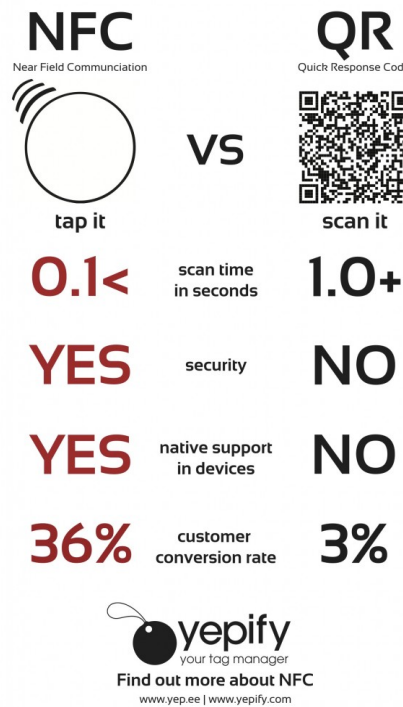


Figure 3. NFC vs QR [8]



After viewing the previous figures it is quite clear that NFC is the best choice for my project. NFC is better because the QR tag would take away from the aesthetic qualities of the racket as well. We don't have to worry about this with NFC because the tag will be hidden inside of the handle of the racket. Next, we will discuss why the Android platform was chosen over its competitors. The main mobile Operating system platforms that are on the market today are Android, IOS, Blackberry, and Windows. Many factors go into choosing an appropriate software platform.

## Mobile OS Market Share

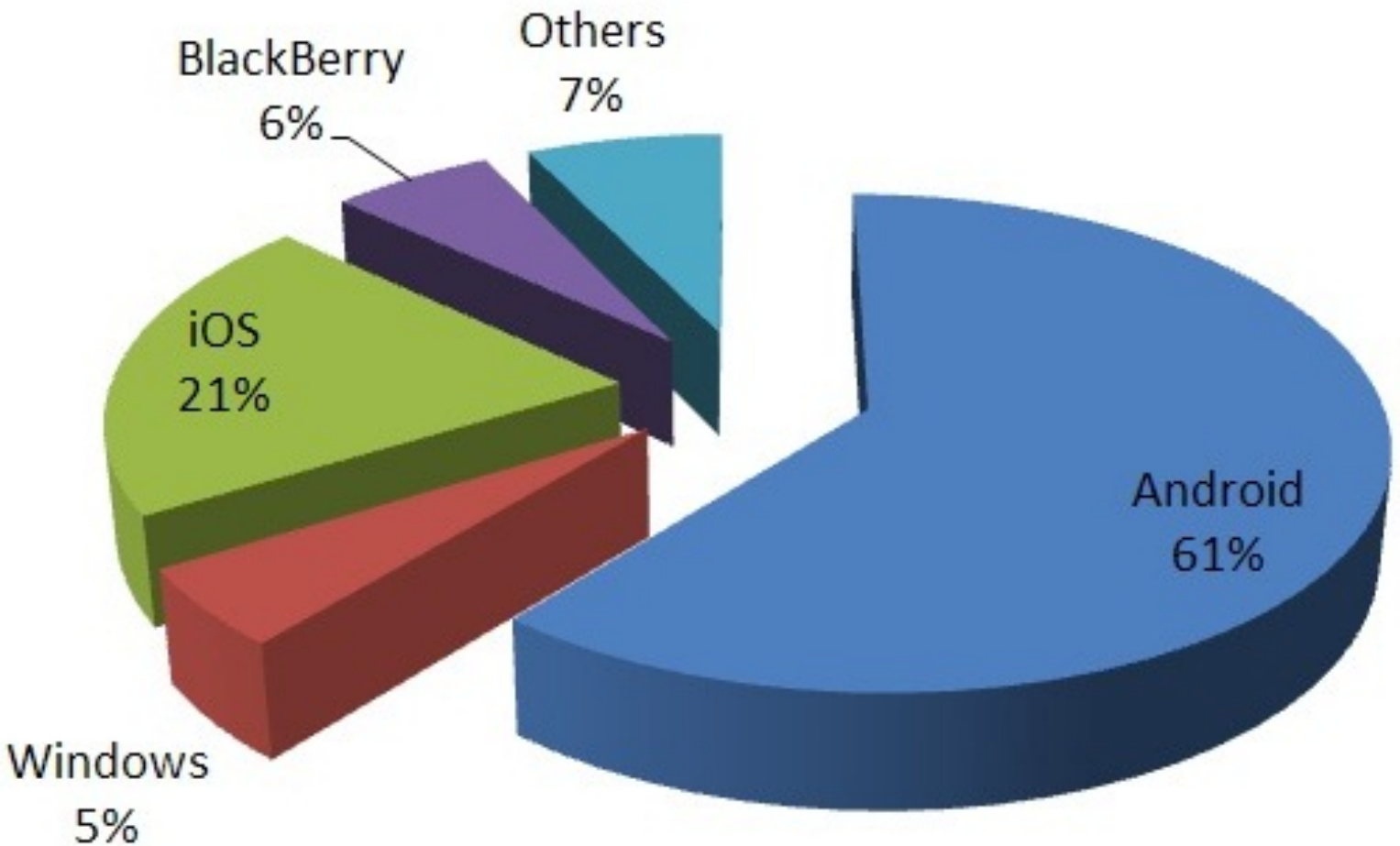


Figure 4. Mobile OS Market Share [11]

The biggest factor in deciding on a mobile operating system would be by determining which system caters to the largest population. When creating software, or any product, best practice is to reach out to the largest customer base possible in order to maximize profits. As seen from Figure 4 above, Android is the largest OS platform by almost three fold. Being the largest platform includes a few nice perks as well as a large support base that can be very helpful when it comes to the actual developmental stages of building the application. Other reasons for not choosing iOS, Blackberry, or Windows, is because of the availability of the devices and their NFC capabilities. Apple recently introduced their newest flagship phones this fall: the iPhone 6 and the iPhone 6 Plus. Both these phones have NFC capabilities, but Apple has not yet made the new technology available to iOS developers. This eliminates iOS from the running. Initially, much consideration was given to going the iOS route since I already own an iPhone and I wouldn't have to purchase another device to develop on. But because there is no access to NFC development in iOS, and my brother owns an Android device, it makes the most sense for me to use Android to develop the application. The final reason Blackberry, or Windows, are not being considered is because of availability and familiarity. There is limited access to a device that runs either OS. Also, previous experience with either device is nonexistent; therefore, Android will be the software platform for the duration of the development process.

Finally there are many options when it comes to developing an Android application. Some potential development platforms are Phone gap, MIT App Inventor, and Android Developer Tools (ADT). The thing that makes Phone gap so appealing is that once you code the application once it can be transferred to any other mobile OS platform which is very nice if you are trying to release an application on multiple platforms. The down side for me though is that it takes more time to initially build the application. Also, most code is written in web-developer languages such as HTML and CSS, which are less familiar platforms. The MIT App inventor is a beautiful web based application design program that uses a graphical user interface. The whole experience is great and easy to use but not as robust as is needed for the scope of this project. That is the downside of most tools that use a GUI, they are easier to use for beginners but they lack features that more advanced users would like access to. The program that was chosen to develop with was ADT and Android Studio, which was developed by Google and has an add-on for eclipse. The reason this program is the most obvious choice is because eclipse and Java are what I have the most coding experience with. Many intro and upper level computer science classes use Java and eclipse on numerous different occasions. The more comfortable while developing and the less to teach my self while completing this project the easier it will be to make the best application possible.

## Preliminary Proposed Design

NFC has been chosen as the medium for identifying each racket. When the racket technician receives a new racket, they will place a new NFC tag inside of the butt cap of the tennis racket. They will then use the Android device with the application installed to setup the new tag. The following sequence of Figures gives a visual representation of the design choices made and how they will be implemented.



Figure 5. NFC Tags



Figure 6. Tennis Racket with NFC Tag installed

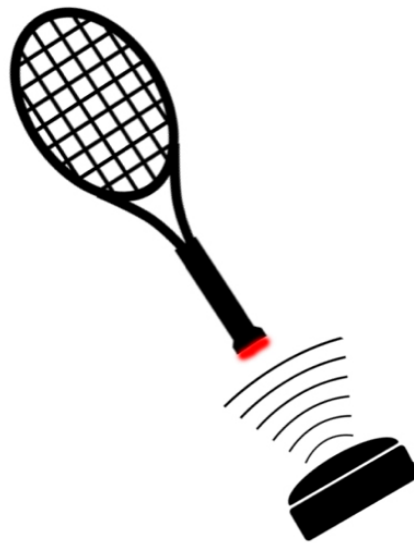


Figure 7. Scan Tag Demonstration



**Figure 8. Live Scan Tag Demonstration**

The figures above do a good job of capturing a visual of what the racket technician will physically be doing when installing and scanning a new tag. All other actions will take place in software.

When it comes to software, as stated in the design specifications section, aesthetics and branding of a product are very important. Out sourced portions of this task like graphic design were handed off to a friend of mine, Caroline Brustowicz,

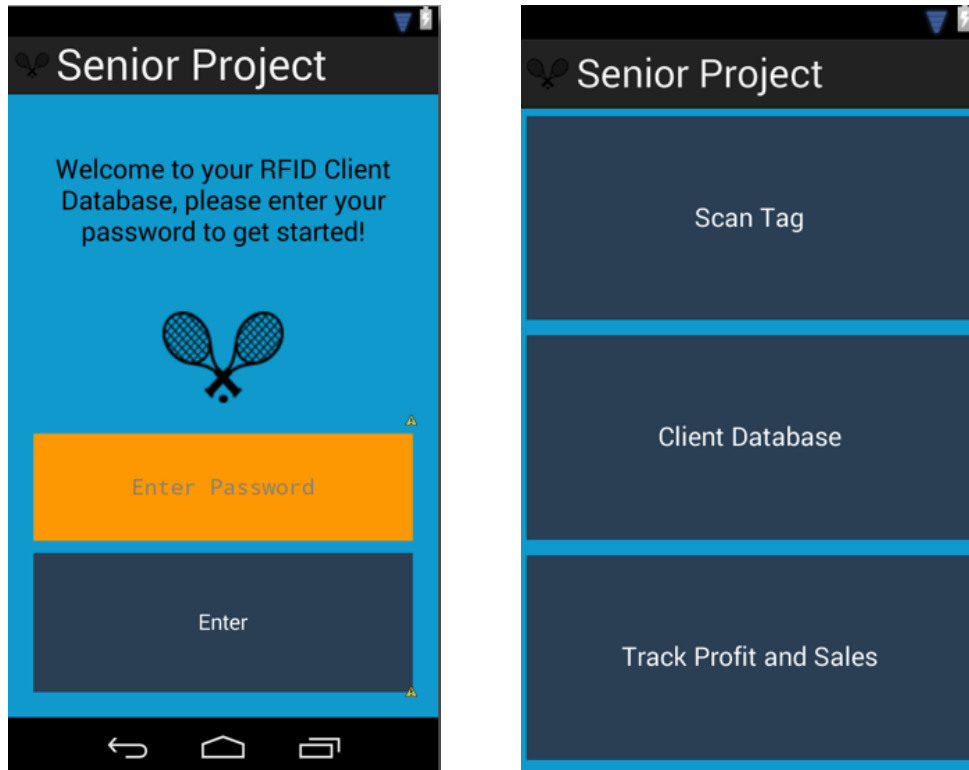
because her digital design skills are best for this task. Together we brainstormed ideas for the a logo. The following figure is a draft of what the logo looks like thus far.



Figure 9. Tracket Logo - Caroline Brustowicz

After establishing a logo, the next most important thing is establishing a first impression. A login and a home screen were created to increase security and to give the user a sense of personalization. The login screen is simple but it gets the point across – it opens with a welcome and prompts the user for a password. If the correct password is entered, the user is brought to the home screen. Currently the password has been hardcoded but later on, the app will have a way for the user to change their

password. Once you are at the home screen, the three main options are displayed equally. The following Figure 10 demonstrates what the login screen and the home screen will look like.



**Figure 10. Login and Home Screens**

The scan tag will take the user to a simple screen with Figure 7 displayed on it informing the user to scan the tennis racket. Once the scan is complete the user will either be taken to the customers file or prompted to set up a new tag. The Client Database button will display a list of all the customers currently stored in the database. At the top of that screen there will also be a search client and an “add new client” button. Finally, the Track Profit and sales button will take the user to a new screen that



will neatly display the users data. The data displayed will be anything that the program has been tracking most importantly, the profit margins, money spent, money received, and any other pertinent pieces of information that the user would like to track.

## **Testing**

Once the prototype for the project has been designed and created and preliminary testing has been completed, the plan is testing it in the setting that it was designed for. The plan is to install the application on my brothers Galaxy S3 and giving him a set of tags to test out the prototype system with. A larger scale prototype test would be to give a few other people that string rackets for clients a chance to use the product, specifically Players Choice Racquet Sports in Schenectady, and the offer to let them test it free of charge if they provided me with detailed feedback on the product's performance in the field. General testing procedures for the project will include the following set of steps:

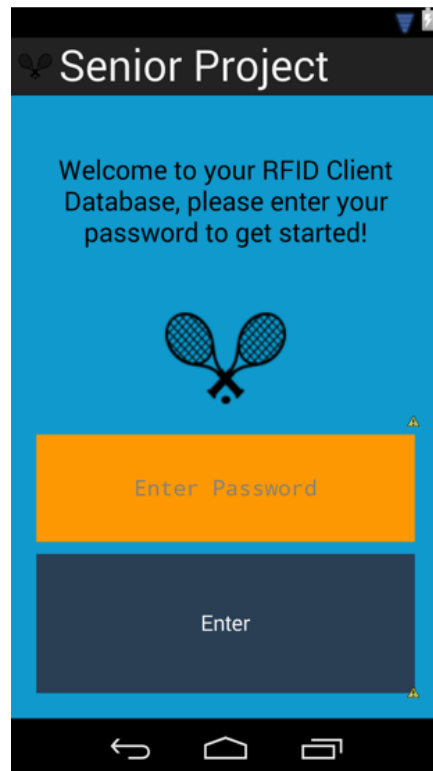
- 1) Write a new serial number to a NFC tag and place it inside the handle of the racket.
- 2) Input the racquet owner's information into a contact and save the data under the serial number.
- 3) Test the reading capabilities to see if the information just entered shows up when scanned.
- 4) See what interesting facts can be gleaned from inspecting the data produced by the application to see if we can ultimately make the business more profitable.

## Final Design and Implementation

This section will get into the bulk of the actual project and how it was done. Each individual screen is called an Android Activity, and each activity has a Java class and an XML file that are running behind the scenes to give the user output and to deal with the user's input. There will be screenshots of the important code displayed in the appendix. As far as how the application works underneath the hood, it starts with the manifest file. The manifest file is one of the first things that is created when you begin creating an Android application. The manifest gives an outline of the app to the device when the app opens up. The manifest contains information such as what pages are in the application, what pages are connected to other pages and it also stores the hierarchy of the application. Certain general specifications are stored in the manifest as well, including what attributes each activity will have. The manifest file is an integral part of the application and the app cannot work without it. When the manifest file is first accessed by the device, the phone will follow the directions given by the manifest, which in the case of my app, the first thing that it will open is the splash screen.

Beginning with the splash screen, the app came to life. Once the manifest gives the directions to open the splash screen, the XML file that stores the activity's layout is opened, the directions given in that file are followed and the initial screen is set up. Once the layout of the activity is established the Java class that deals with the back-end of the activity is run. The Java class for this activity is equipped with listeners, so when a user presses the button to enter the password, the method in the Java class will then compare the string value that was input by the user to the string value that has

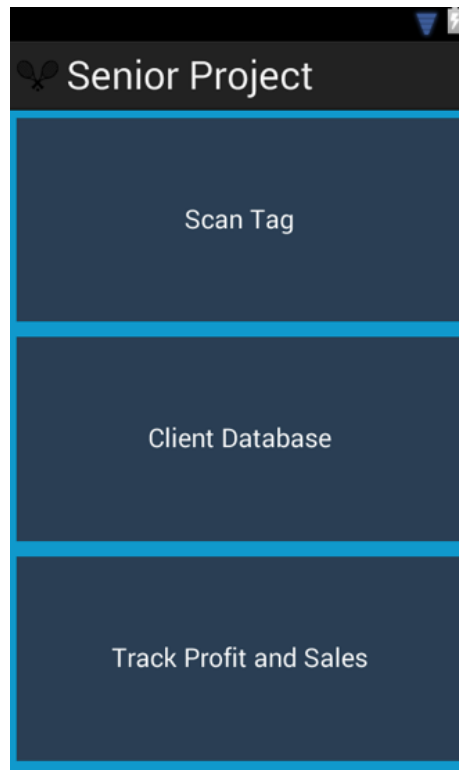
been deemed the valid password for entry. Below is a screen shot of the layout for the splash screen.



**Figure 11. Login screen**

The main purpose of this screen is to introduce the user to the application and to make sure that unauthorized users cannot gain access to the application. The reason for this is, in future versions of the application the app may store personal contact, billing, and other sensitive information that the user may want to keep private. The initial login password is 0001, which the user can change later on. The password checking method is setup so if the password is correct it will pull up the main menu, if the password is incorrect, the string that was put in will be erased and the user will be prompted to try again.

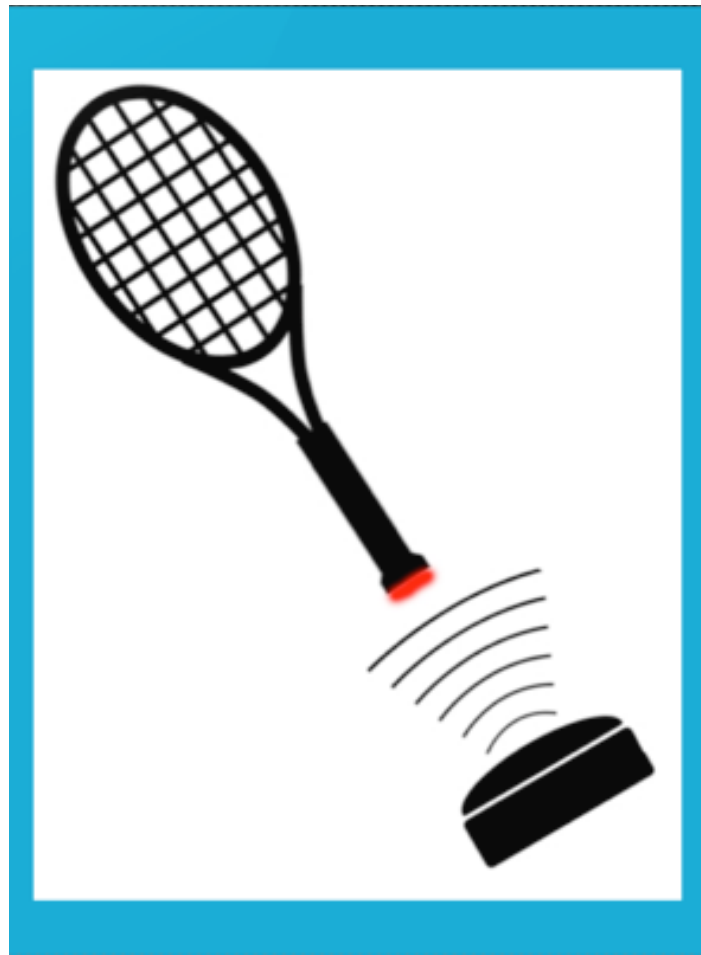
Once the correct password has been entered the user will be brought to the main menu. The main menu is pretty self explanatory, there are three main options that the user can choose and in the next version of the application the user should be able to select a change password option as well.



**Figure 12. Main Menu**

Above we see the three options available to the user in the main menu; Scan Tag, Client Database, and Track Profit and Sales. We will navigate this menu and go through each option to give a full understanding of every capability of this application. Since you already have a good understanding of what the app is doing, we will start at the top with the scan tag option.

The following Figure is a Screen shot of the activity Scan Tag that can be selected from the main menu.

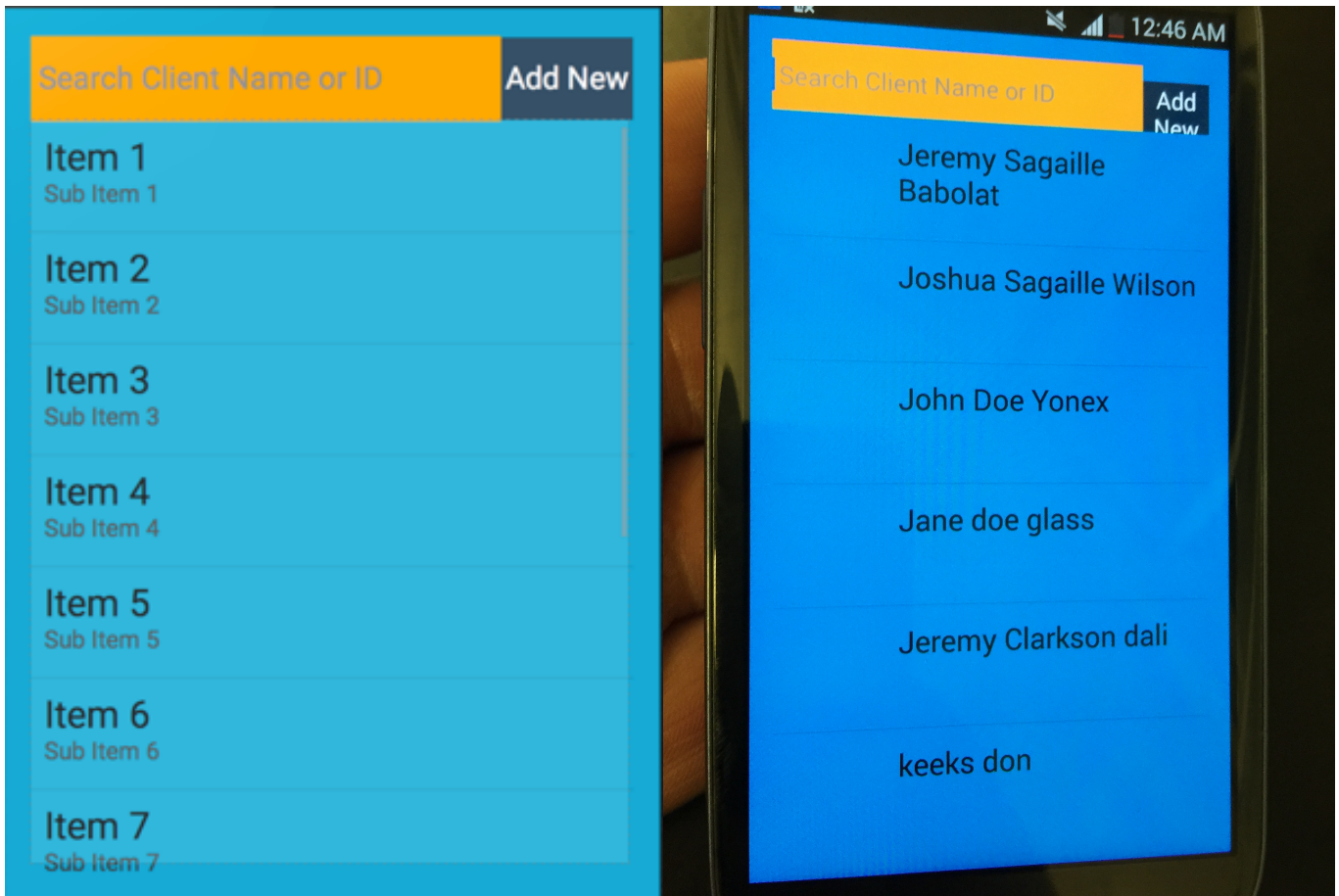


**Figure 13. Scan Tag**

The scan tag screen is a very important part of the app because it gives the user the ability to quickly scan the tag and pull up the the corresponding client. While there is a specific activity for scanning the tag, once the app is opened the NFC card is enabled which means if you scan an existing tag in another activity, the client data will still be displayed. Currently, because of unforeseen difficulties, while the app does scan an active tag, it does not yet pull up the client data. This is the first thing that will be fixed

in version 2 of the application because a large part of the appeal to use this app is that you don't need to search a database and that the phone would be able to know which client the racket belonged to just by being close to it. Now that we have talked about scanning a tag and opening up a client. Lets talk about the client data base and creating new clients, which is the next option on the main menu.

When the client database option from the main menu is selected the database activity is opened up next. Below is a screen shot and picture of that activity.



**Figure 14. Client Database**

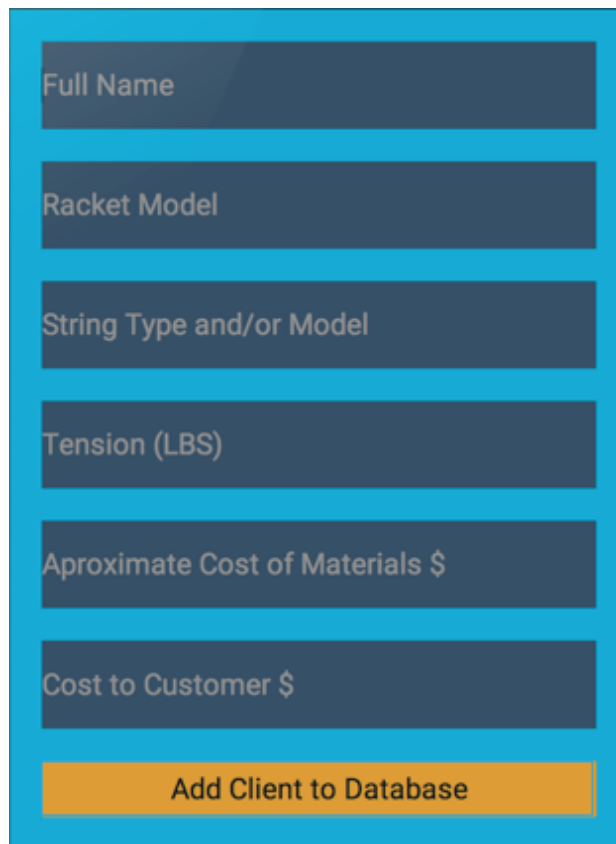
Once you have opened the client database activity there are a few options. First the majority of the layout is taken up by the list view that will be populated from the SQLite database running in the background in a separate Java class. Other attributes on the page include a search bar and a button to add a new clients. SQLite is a very important part of this application, it is what enables the app to be useful. When the app is first opened an SQLite database is created that has columns to store relevant information on clients such as:

- Client ID
- Name
- Racket Type
- String Type
- String Tension
- Most recent overhead paid
- Most recent cost to customer
- Date and Time last strung
- Total overhead paid
- Total cost to customer

```
private static final String TABLE_CREATE =
    "CREATE TABLE " + TABLE_CLIENTS + " (" +
        COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
        COLUMN_NAME + " TEXT, " +
        COLUMN_RACKET + " TEXT, " +
        COLUMN_STRINGTYPE + " TEXT, " +
        COLUMN_TENSION + " NUMERIC, " +
        COLUMN_OVERHEAD + " NUMERIC, " +
        COLUMN_PRICE + " NUMERIC, " +
        COLUMN_DATE + " TEXT, " +
        COLUMN_TIMESSTRUNG + " NUMERIC, " +
        COLUMN_TOTOVERHEAD + " NUMERIC, " +
        COLUMN_GROSSPROFIT + " NUMERIC " +
    ")";
```

**Figure 15. SQLite Database Setup**

Figure 15 displayed previously is a screen shot of the code that sets up the data base. Now that we have discussed how the database works, lets move on to how we would add a new client to the database. Below is a screenshot of an activity that would take in the information to add a new client.



The screenshot shows a form with a blue border. It contains six text input fields stacked vertically, each with a light blue placeholder text. The fields are labeled: "Full Name", "Racket Model", "String Type and/or Model", "Tension (LBS)", "Aproximate Cost of Materials \$", and "Cost to Customer \$". Below these fields is an orange button with the text "Add Client to Database".

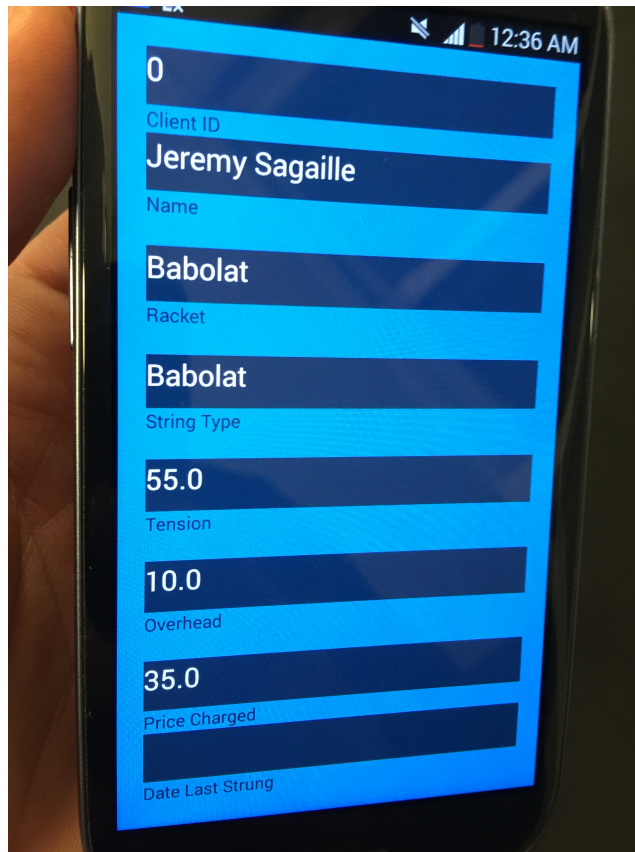
**Figure 16. New Client**

When we want to go ahead and add a new client we would fill out the activity shown above. After all of the information is input and the user presses the “add client to database” button, a new row is created inside of the SQLite database and the columns are filled with the data given above. The method that creates a new client also adds in some data automatically, like the date, client ID the totals for overhead and gross profit. Once the client has been added, a message is sent to the screen telling the user that



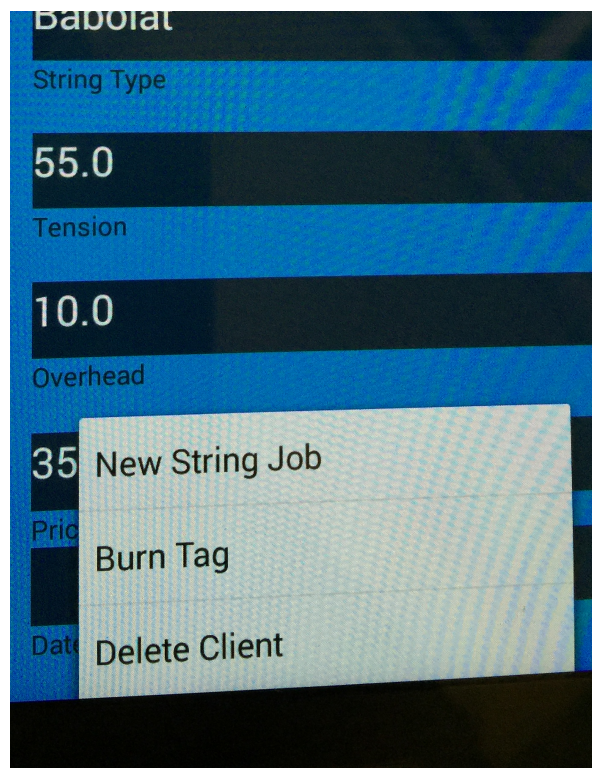
the client has been successfully added into the database and also says the the client ID number. The display of the client ID number was done on purpose so that when the tag was originally written, the racket technician could write the ID number on the tag as a backup in case the tag malfunctioned for some reason. If the tag was unreadable the ID could be physically read off the tag and then searched using the search bar provided in the client database activity.

After the client has been added to the database the next logical step is to view the client that we just created. Below in Figure 17 is a picture of what it looks like when we select a client from the list view in the Client Database activity.



**Figure 17. Client View**

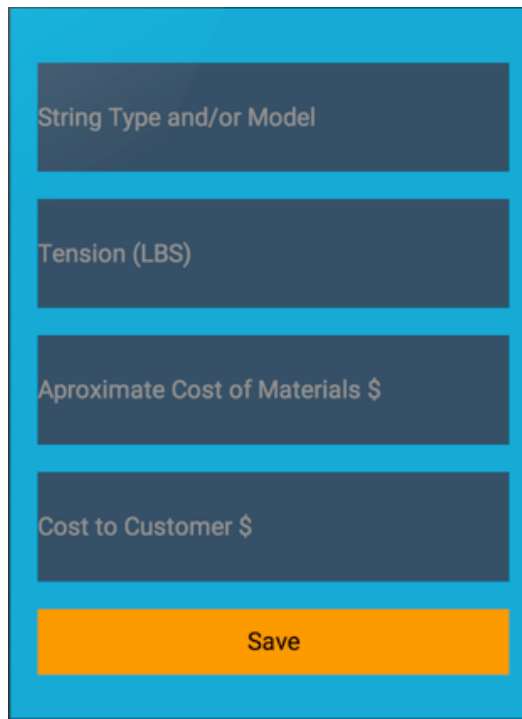
As you can see above the contents of the client view is essentially the data that was input from the add client activity. The only thing that is different is that the client ID and the date are shown. Currently, the date in the previous figure is not visible due to an error in the code. The next step is to be able to burn the client ID to a tag and edit the data. While viewing the client that you would like to manipulate, if you hit the android menu soft key to the left of the home button the options in the figure below appear.



**Figure 18. Client View Menu**

The three options displayed are New String Job, Burn Tag, and Delete Client. The first option, New String Job, allows the user to update the information originally entered for the client, including the string type, tension, the overhead to complete the job, and the price charged to the customer. Below in the next figure is another screen

shot from the application that shows the activity that comes up when the user chooses New String Job.



The screenshot shows a mobile application interface for creating a new string job. It features a blue background with four dark blue input fields stacked vertically. The labels for the fields are: 'String Type and/or Model', 'Tension (LBS)', 'Aproximate Cost of Materials \$', and 'Cost to Customer \$'. At the bottom of the form is an orange button labeled 'Save'.

**Figure 19. New String Job**

As you can see in the figure above, the user is prompted to enter the new information. Once the user hits save, the row in the SQLite database that is holding the client's data is updated with the new information. In the background of the app, the columns with the client's date last strung, total overhead, and gross profit are also updated accordingly.

The next option on the menu from the client view is to burn a tag. This option would be used right after a client is created and could also be used later on if a client's tag were damaged or lost. The screen shot is not included because it is a very simple

activity that pops up with a message instructing the user to touch a tag to the back of the device. When the activity is initially opened, the Java class in the background enables the NFC module in the device and when the device comes within close proximity of a blank tag, a string representation of the client ID number is written on to the tag. When the tag is written, a message is broadcasted to the screen informing the user that the tag has been successfully written. Another message also comes up displaying the client's ID again, so it can be written down on the tag as a back up which was previously described.

The last option available in the client view menu is the option to delete a client. When the option is selected it removes that client from the database. With all of the possibilities inside of the Client Database option from the main menu having been explored we will move to the final option on the main menu which is Track Profit and Sales. Unfortunately there was not enough time to finish this portion of the application but below are some of the things that will be displayed when this piece of the app is finished.

- Total number of clients
- Total overhead
- Gross Profit
- Net Profit
- Average number of string jobs per month
- Average net profit per month

## Performance Estimates and Results

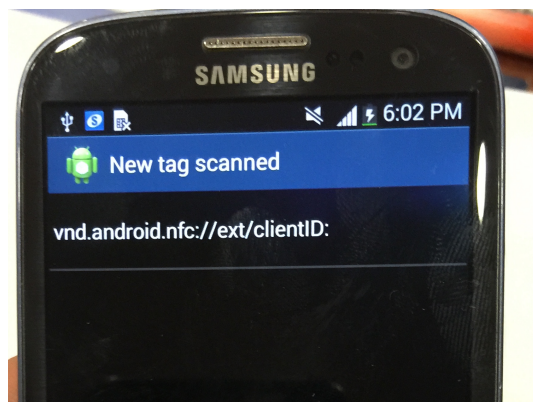
Now that the functionalities included in this application have been described next we will focus on how well the goals were achieved and how successful the project was. Original goals were:

- The implanted NFC Tag must be undetectable by the tennis player.
- The device must easily read the tag's serial number.
- The application must be easy to use.
- The application should be able to create a new tag.
- If a tag is somehow lost or damaged then the old serial number can be written on to a new tag.
- The app must track sales and profits.
- The app must track materials used.
- The app must track customer preferences on racquet jobs along with a history of jobs done.
- Include a way of backing up the data.
- Secure the data.

While success was achieved in most of these goals, not all of them were completed fully due to lack of time at the end of the project. Next we will go through the goals and discuss how they were accomplished and if not, why they weren't and how they will be accomplished in future versions of the project.

The first thing on the list is to ensure that the client will not be disturbed by the tag. As we discussed earlier this will be a non-issue because the NFC tags that are

being used in this project are paper thin and weigh less than 5 grams. The client will never notice the difference. Also, the device must easily read the tag. This was described as a goal because originally it was not certain whether the handle of a tennis racket would produce any interference. It turns out that after test scanning a tag from several different racket models there is no interference because the handles of rackets are generally made from plastic or carbon fiber. The application had to be easy to use as well. This goal was accomplished, because the app is very straightforward. Options are logically placed and effective. Next the app needs to be able to create a new tag and then be able to read the tag. This is one of the goals that was not completed in the time allocated for this project. The app currently will create and read a new tag, and if the tag gets lost then you have an option to burn another tag. The only portion of this process that does not work is when a tag is scanned, the ID is not correctly read off of the tag. Since the ID number is not read, the data pertaining to the ID number cannot be retrieved either. The figure below shows the activity that appears when a tag has been read. As you can see the client ID is not displayed.



**Figure 20. Failed Tag Scan**

The next goal is that the app was to track profit sales and materials used.

This was not completed either. Once the rest of the app is set up, however, this should be one of the easiest parts to finish. All of the data will either be drawn straight from the database or it will be manipulated with basic math. Another thing that would be good to add in here eventually would be some basic histograms demonstrating how much money is being made month to month. Next the app must track customer preferences on racquet jobs along with a history of jobs done. This goal was completed and these preferences will be stored in the SQLite database. As far as backing up the data stored in the app, this may come in a later release as well. It also may be better to possibly transfer the SQLite database into a SQL database. The reason being that SQL is server based so not only could we back up data on a server, we could potentially share databases between devices. One of my final goals was to make sure there was some sort of security in place. This goal was accomplished by simply adding password protection to the splash screen.

Even though the entire project was not completed to the previous specifications that I set for myself. I learned a lot about how hard it is to accurately predict how long it takes to do something you haven't done before. This funnels straight into the next part which discusses the schedule that was set out at the beginning of the project.

## Production Schedule

At the beginning of this ten week term I met with my advisor (Professor Traver) to discuss a plan of attack for the term. My goal was to have all of my coding and testing on the software done by the end of the 8th week because that was when the final presentations were. At the beginning, we agreed on splitting the work into several sections and then having tasks due every week to keep me on track. Since starting the app development before the term began, I had a little bit of a head start on the work which was nice. The framework for the most part, had already been laid out.

We originally agreed that I would split the 8 weeks into two parts. The first 4 weeks would be finishing the initial framework and layout then I would build and test the SQLite database. The second 4 weeks would be dedicated to finishing the NFC portion of the app and testing the app thoroughly. While this schedule seems straight forward I learned throughout the term several times that things rarely go as planned. Sometimes things go quicker, such as when I was setting up the layout, but the majority of the time they do not. When creating the SQLite data base it took until the end of week 5 to almost finish it and then later on in the term I had to go back again and revise the data base because there were things that I had forgotten to include.

When I finally started working on the NFC portion, it was going really slow and was not nearly as straight forward as I had previously anticipated. On top of that, I had



put the SQLite data base into place, but I didn't really know how to use it. Figuring out how to pull specific pieces of data was not as easy as it first seemed either and I ended up losing another week here figuring all of this out. Finally I began working on the NFC but I did not finish this portion because I ran out of time.

If I were to do this project again I think I would have tried to start a little bit earlier, because the project that I chose was a very ambitious amount to get done in 8 weeks. I also think if I had had a mentor that knew more about android app development the process could have been expedited. At the same time, I think that since I had to do everything on my own, the experience proved to be quite valuable. Since I had no one to reach out to for help, I had to rely on myself to trouble shoot all problems and issues which is a great skill to have.

## Cost Analysis

The cost analysis is the simplest part of this report because because there were no direct costs . First lets discuss the hardware. The device that I developed the application for and tested on was a Galaxy S III and it was an old phone that belonged to my brother. The only other cost was the NFC tags which I bought off of eBay. The tags are cheap, they only cost about a dollar a piece and if you buy them in bulk they were even cheaper.

All of the software that was used to develop the application was free. When I first began developing the app I was using the eclipse IDE for Java with an add on built by google to make it easier to develop Android apps. About halfway through the project google deployed their own IDE called android studio which is also free so I began using that. Android Studio was easier to use and much faster so it was a win win for me.

## User's Manual

Before you can begin using this prototype you must first be sure that you have assembled the appropriate parts. First you will need an NFC enabled Android device. Next you will need several NFC tags, and finally a copy of the compiled app code. First you should install the application and then make sure you have your NFC tags on hand. Once the application is installed, open it and navigate to the main menu, to do this you will enter the password 0001 and press enter. Once you are at the main menu, chances are that you would like to start by adding a client because otherwise you cannot really do much.

To add a client we must begin by pressing the 2nd option on the main menu page, Client Database. Once in the client database page at the top right we will select the add new button which will bring us to a page where we can add the appropriate data for a new client. Once the data has been entered press the button at the bottom of the page and this will add all of the client's data into the database. Once the Client is added we will be directed back to the Client Database page where we can go ahead and manipulate the client. If we select the client from the list view that we just created, the client's information will be pulled back up and displayed. While viewing the client, if we hit the android soft key menu button, three options are presented, burn tag, new string job, and delete client. selecting burn tag will give you the option to burn a tag with the clients ID number on it which can then be looked up later on and retrieve client

data automatically by being scanned. The new string job option allows the user to update the client with recent data from a new string job. Finally, delete client does just that, it removes the current client from the data base.

The top option on the main menu, Scan Tag, is self explanatory. If you select this option the you will be able to scan a previously burned tag and lookup the corresponding client data. The last option available in the main menu is where you can go to view basic statistics on how much money you are spending, making, number of clients and other useful data.

## **Discussion, Conclusions, and Recommendations**

The reason I got into this project was because I saw a process and thought, how can I make this better? My brother was making pretty good money sure, but how much, and what could he do to possibly push the boundaries of how much he was making. The easiest was to analyze the problem from a statistical point of view, and in order to do this we must begin by collecting data. An easier way to do this could have been just writing values down, but why not use a simple combination of hardware and software to help automate the process and drive up efficiency and productivity? This is where my idea was born.

I think one of the biggest things that I learned while doing this project is that things always take significantly longer than you think they will. For example I first came up with this idea in 2012 and it wasn't till almost two years later that I was able to grasp an opportunity that gave me the ability to finally develop my idea. When I began the senior project track back in the first and second of the three parts, a lot of research was put into the senior project topic. This seemed tedious at the time because I was really excited to start coding. But this research saved me a lot of time in the long run and gave me great insight into good resources later on.

As I went through the steps of developing the software I also learned that you need to adapt. Some of the things that I thought were good ideas in the beginning turned out to be bad ways or incorrect ways of accomplishing things. Also, I feel that I could have probably been a bit more detailed with my initial outline of how I did things because at one point I had been working for so long on being able to display the client data from the database that I totally forgot to implement functionality to be able to add multiple string jobs to a single client. Since this happened when I went into to fix the problem, it was a lot harder to put in the function than if I had built it in when I had first started coding.

I feel that I took a top down approach to developing the application which helped the organization and flow of the whole project. By starting with the splash screen and making sure that the overall basic structure of the app was working I was able to avoid more troubleshooting at the end. I also didn't try to develop the database and the NFC parts at the same time, I split them up and did one then the other. At first I was tempted to do them together because the pieces do go hand in hand, but after careful consideration I decided to do them separately. Doing the parts separately was a good idea because while the parts are related, the software that is required to make the pieces run is very different and the things that you need to learn are both abstract and quite difficult.

In conclusion, this project helped me learn a great deal about myself, what I am capable of and what the professional work world is like. This project showed that I am able to persevere through long nights of working and eventually move on to figure out how to trouble shoot issues on things that I taught myself how to do. This is a great life skill and definitely an experience that I can share with other people. Even though I was not able to finish the project in the time allowed, I think that I have made it past the hardest parts of the application. I am eager to finish the application completely and possibly add some additional features to push the boundaries of my own knowledge base and what I think I am capable of accomplishing.

## References

- [1] Douglas J. Thomas, Paul M. Griffin, "Coordinated Supply Chain management," vol. 94, 1996.
- [2] L. M. Katina Michael, "The Pros and Cons of RFID in Supply Chain Management," *Ieee*, vol. 6, 2005.
- [3] Pr. Serge Miranda\*\*, Nicolas Pastorelly\*, "NFC ubiquitous information service prototyping at the University of Nice Sophia Antipolis and multi-mode NFC application proposal", *Ieee*, 2011.
- [4] Vedat Coskun, Kerem Ok, Busra Ozdenizci, "Introduction," in *NFC Application Development for Android* Anonymous 2013, pp. Introduction.
- [5] R. Want, "RFID: A Key to Automating Everything." *Ebscohost*, 2004.
- [6] Tommy Haas. *Making A Racket!*. Available: [http://news.bbc.co.uk/sportacademy/hi/sa/tennis/features/newsid\\_3000000/3000836.stm](http://news.bbc.co.uk/sportacademy/hi/sa/tennis/features/newsid_3000000/3000836.stm).
- [7] (August 21 2014). What makes good code good at INTECOL13. Available: <http://www.software.ac.uk/blog/2013-08-23-what-makes-good-code-good-intecol13>.
- [8] (2014). Near Field Communication (NFC) allows mobile phones to interact with the real world.. Available: <http://www.rjw-tap.com/what-is-near-field-communication/#header>.



[9] (07/03/2013). Visual vs physical proximities. Available: <http://www.yep.ee/industry/visual-vs-physical-proximities/>.

[10] (February 2009). 7 things you should know about... QR Codes. Available: <http://net.educause.edu/ir/library/pdf/ELi7046.pdf>.

[11] (June 7 2012). Research Shows Android Will Dominate Mobile Phone OS Market for Over 5 Years. Available: <http://thedroidguy.com/2012/06/research-shows-android-will-dominate-mobile-phone-os-market-for-over-5-years-24071>.

# Appendices

## Splash Screen Java class

```
package com.example.seniorproject;

import ...

public class Splash extends Activity {
    public final static String EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
        ActionBar actionBar = getActionBar();
        actionBar.hide();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.splash, menu);
        return true;
    }

    /** Called when the user clicks the Send button */
    public void sendMessage(View view) {
        // Intent intent = new Intent(this, DisplayMessageActivity.class);
        // EditText editText = (EditText) findViewById(R.id.edit_message);
        // String message = editText.getText().toString();
        // intent.putExtra(EXTRA_MESSAGE, message);
        // startActivity(intent);

        EditText et = (EditText) findViewById(R.id.edit_message);
        String password = et.getText().toString();
        et.setEditableText().toString();

        if (password.equals("0001")) {
            Intent home = new Intent(this, MainMenu.class);
            startActivity(home);
        } else {
            et.setText("");
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Incorrect Password");
            builder.setMessage("Try Again!");
            builder.setPositiveButton("OK", null);
        }
    }
}
```

## Splash Screen XML Layout Code

```
package com.example.seniorproject;

import ...

public class Splash extends Activity {
    public final static String EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
        ActionBar actionBar = getActionBar();
        actionBar.hide();
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.splash, menu);
        return true;
    }

    /** Called when the user clicks the Send button */
    public void sendMessage(View view) {
        // Intent intent = new Intent(this, DisplayMessageActivity.class);
        // EditText editText = (EditText) findViewById(R.id.edit_message);
        // String message = editText.getText().toString();
        // intent.putExtra(EXTRA_MESSAGE, message);
        // startActivity(intent);

        EditText et = (EditText) findViewById(R.id.edit_message);
        String password = et.getText().toString();
        et.setEditableText().toString();

        if (password.equals("0001")) {
            Intent home = new Intent(this, MainMenu.class);
            startActivity(home);
        } else {
            et.setText("");
            AlertDialog.Builder builder = new AlertDialog.Builder(this);
            builder.setTitle("Incorrect Password");
            builder.setMessage("Try Again!");
            builder.setPositiveButton("OK", null);
        }
    }
}
```

## SQLite Setup code methods

```

public Client create(Client client){
    ContentValues values = new ContentValues();
    values.put(ClientsDBOpenHelper.COLUMN_NAME,client.getName());
    values.put(ClientsDBOpenHelper.COLUMN_RACKET,client.getRacket());
    values.put(ClientsDBOpenHelper.COLUMN_STRINGTYPE,client.getStringType());
    values.put(ClientsDBOpenHelper.COLUMN_TENSION,client.getTension());
    values.put(ClientsDBOpenHelper.COLUMN_OVERHEAD,client.getOverhead());
    values.put(ClientsDBOpenHelper.COLUMN_PRICE,client.getPrice());
    values.put(ClientsDBOpenHelper.COLUMN_DATE,client.getDate());
    values.put(ClientsDBOpenHelper.COLUMN_TIMESSTRUNG,client.getTimesStrung());
    values.put(ClientsDBOpenHelper.COLUMN_TOTOVERHEAD,client.getTotalOverhead());
    values.put(ClientsDBOpenHelper.COLUMN_GROSSPROFIT,client.getGrossProfit());
    long insertID = allClients.insert(ClientsDBOpenHelper.TABLE_CLIENTS,null,values);
    client.setId(insertID);
    return client;
}

```

```

public Client update(Client client){
    ContentValues values = new ContentValues();
    values.put(ClientsDBOpenHelper.COLUMN_STRINGTYPE,client.getStringType());
    values.put(ClientsDBOpenHelper.COLUMN_TENSION,client.getTension());
    values.put(ClientsDBOpenHelper.COLUMN_OVERHEAD,client.getOverhead());
    values.put(ClientsDBOpenHelper.COLUMN_PRICE,client.getPrice());
    values.put(ClientsDBOpenHelper.COLUMN_DATE,client.getDate());
    values.put(ClientsDBOpenHelper.COLUMN_TIMESSTRUNG,client.getTimesStrung());
    values.put(ClientsDBOpenHelper.COLUMN_TOTOVERHEAD,client.getTotalOverhead());
    values.put(ClientsDBOpenHelper.COLUMN_GROSSPROFIT,client.getGrossProfit());
    long insertID = allClients.insert(ClientsDBOpenHelper.TABLE_CLIENTS,null,values);
    client.setId(insertID);
    return client;
}

```

```

public List<Client> findAll(){
    List<Client> clients = new ArrayList<>();

    Cursor cursor = allClients.query(ClientsDBOpenHelper.TABLE_CLIENTS, allColumns,
        null, null, null, null, null);

    if(cursor.getCount()>0){
        while(cursor.moveToNext()){
            Client client = new Client();
            client.setId(cursor.getLong(cursor.getColumnIndex(ClientsDBOpenHelper.COLUMN_ID)));
            client.setName(cursor.getString(cursor.getColumnIndex(ClientsDBOpenHelper.COLUMN_NAME)));
            client.setRacket(cursor.getString(cursor.getColumnIndex(ClientsDBOpenHelper.COLUMN_RACKET)));
            client.setStringType(cursor.getString(cursor.getColumnIndex(ClientsDBOpenHelper.COLUMN_STRINGTYPE)));
            client.setTension(cursor.getDouble(cursor.getColumnIndex(ClientsDBOpenHelper.COLUMN_TENSION)));
            client.setOverhead(cursor.getDouble(cursor.getColumnIndex(ClientsDBOpenHelper.COLUMN_OVERHEAD)));
            client.setPrice(cursor.getDouble(cursor.getColumnIndex(ClientsDBOpenHelper.COLUMN_PRICE)));
            clients.add(client);
        }
    }
    return clients;
}

```

## Burn tag method

```

boolean writeNdefMessageToTag(NdefMessage message, Tag detectedTag) {
    int size = message.toByteArray().length;
    try {
        Ndef ndef = Ndef.get(detectedTag);
        if (ndef != null) {
            ndef.connect();

            if (!ndef.isWritable()) {
                Toast.makeText(this, "Tag is read-only.", Toast.LENGTH_SHORT).show();
                return false;
            }
            if (ndef.getMaxSize() < size) {
                Toast.makeText(this, "The data cannot written to tag, Tag capacity is " + ndef.getMaxSize()
                    + " bytes, message is " + size + " bytes.", Toast.LENGTH_SHORT).show();
                return false;
            }

            ndef.writeNdefMessage(message);
            ndef.close();
            Toast.makeText(this, "Message is written tag.", Toast.LENGTH_SHORT).show();
            return true;
        } else {
            NdefFormatable ndefFormat = NdefFormatable.get(detectedTag);
            if (ndefFormat != null) {
                try {
                    ndefFormat.connect();
                    ndefFormat.format(message);
                    ndefFormat.close();
                    Toast.makeText(this, "The data is written to the tag ", Toast.LENGTH_SHORT).show();
                    return true;
                } catch (IOException e) {
                    Toast.makeText(this, "Failed to format tag", Toast.LENGTH_SHORT).show();
                    return false;
                }
            } else {
                Toast.makeText(this, "NDEF is not supported", Toast.LENGTH_SHORT).show();
                return false;
            }
        }
    } catch (Exception e) {
        Toast.makeText(this, "Write operation is failed", Toast.LENGTH_SHORT).show();
    }
    return false;
}

```

## Client Object Java class

```

public class Client {
    private long id;
    private String name;
    private String racket;
    private String stringType;
    private double tension;
    private double overhead;
    private double price;
    private String date;
    private double timesStrung;
    private double totalOverhead;
    private double grossProfit;

    public Client(){
        id = 0;
        name = "TEST";
        racket = "TEST";
        stringType = "Test";
        tension = 55;
        overhead = 10;
        price = 35;
    }

    public Client( String NAME, String RACKET, String STRINGTYPE, double TENSION, double OVERHEAD,
        double PRICE, String DATE, double strung, double TO, double GP){
        //id = ID;
        name = NAME;
        racket = RACKET;
        stringType = STRINGTYPE;
        tension = TENSION;
        overhead = OVERHEAD;
        price = PRICE;
        date = DATE;
        timesStrung = strung;
        totalOverhead = TO;
        grossProfit = GP;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getRacket() { return racket; }
    public void setRacket(String racket) { this.racket = racket; }
    public String getStringType() { return stringType; }
    public void setStringType(String stringType) { this.stringType = stringType; }
    public double getTension() { return tension; }
    public void setTension(double tension) { this.tension = tension; }
    public double getOverhead() { return overhead; }
    public void setOverhead(double overhead) { this.overhead = overhead; }
    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }
    public String getDate() { return date; }
    public void setDate(String date) { this.date = date; }
    public double getTimesStrung() { return timesStrung; }
    public void setTimesStrung(double ts) { this.timesStrung = ts; }
    public double getTotalOverhead() { return totalOverhead; }
    public void setTotalOverhead(double to) { this.totalOverhead = to; }
    public double getGrossProfit() { return grossProfit; }
    public void setGrossProfit(double gp) { this.grossProfit = gp; }

    @Override
    public String toString() {
        NumberFormat nf = NumberFormat.getCurrencyInstance();
        return name + " " + racket;
    }
}

```

## Displaying a client Java class

```

ClientDataSource datasource;

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

    ActionBar actionBar = getActionBar();
    actionBar.hide();

    datasource = new ClientDataSource(this);
    datasource.open();

    Bundle extras = getIntent().getExtras();
    Long clientID = null;
    if (extras != null) {
        clientID = extras.getLong("clientID");
    }

    //Get Client
    Client client = datasource.queryClient(clientID);
    TextView i =(TextView)findViewById(R.id.CLIENTID);
    i.setText("" + clientID);

    TextView n =(TextView)findViewById(R.id.clientName);
    n.setText(client.getName());

    TextView r =(TextView)findViewById(R.id.racketModel);
    r.setText(client.getRacket());

    TextView s =(TextView)findViewById(R.id.StringType);
    s.setText(client.getStringType());

    TextView t =(TextView)findViewById(R.id.tension);
    t.setText("" + client.getTension());

    TextView o =(TextView)findViewById(R.id.myCost);
    o.setText("" + client.getOverhead());

    TextView p =(TextView)findViewById(R.id.salePrice);
    p.setText("" + client.getPrice());

    TextView d =(TextView)findViewById(R.id.DATELASTSTRUNG);
    //d.setText(client.getDate().toString());
    //Toast.makeText(this, client.getDate().toString(),Toast.LENGTH_LONG).show();
}

```