CS4624 Multimedia/Hypertext

Spring 2015

21st Century Inventory App

Brandon Dean, Elliot Garner, Brannon Mason

Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

Date: May 5, 2015
Emails: deanbr@vt.edu,
elliot94@vt.edu,
brannon1@vt.edu
Submitted to: Prof. Edward Fox

Executive Summary

Currently, Network Infrastructure & Services (NI&S) takes inventory of equipment assigned to employees (computers, laptops, tablets, tools) and sends reports of higher value items to the Controller's Office. All items have a VT tag number and a CNS number, which can currently only be matched up via an Oracle Forms interface. An inventory clerk must personally verify the existence and location of each piece of equipment. An improvement would be an app that scans an inventory number or bar code and the GPS location where it is scanned and the custodian of that equipment. This data could then be uploaded to a more accessible Google spreadsheet or similar web-based searchable table.

The 21st Century Inventory app aims to solve this problem by employing barcode scanning technology integrated into a mobile app which would then send the accompanying asset ID to a CSV formatted output file. By directly tying a product's asset ID to the user and their information, along with having the capability to scan a product's barcode to simplify inventory lookup, saving product information to a CSV file, and giving the user the ability to edit the current information of a product in the application, we are providing a significant upgrade to a system that currently solely relies on an Oracle Forms interface.

Table of Contents

1. Problem Specification

1.1 Purpose

The current system in use by the CNS department at Virginia Tech lacks a central-standard identification unit, leading to difficulties for staff members to have an accessible way to track the communication inventory. For this reason, the 21st-Inventory will mobilize and centralize the tracking of the inventory through the development of a mobile application.

A motif for the 21st-Inventory project is the potential for it to be a prototype for future updates to the Oracle Docs system employed by the CNS department at Virginia Tech. Documentation, a friendly user interface, and ease-of-use will be key factors in trying to convince the department to upgrade their system.

1.2 Description of App's Expected Functionality

Employees of the CNS department have expressed their difficulty in navigating the archaic Oracle Docs system. The current set-up requires entering different identification values to track down a single object in the system. Further, employees manually have to enter information about the items that they add into the system, increasing the chance of random error.

21st-Inventory will act as a buffer between the employee and the Oracle Docs system by accessing a central .csv file to acquire and store information. It should be noted that the employee will be able to take a picture of a bar-code that will update the .csv file with the real-time information rather than solely having to enter the information manually. Expected gains from implementing 21st-Inventory include an increase in the workflow and a decrease in the expected cost from untracked inventory.

To improve performance costs, the current revision for the inventory list will be downloaded to the mobile device in use by a CNS employee, where it will be parsed to allow for easy management if the user needs to constantly update the list. After finishing the necessary updates, the user can then upload the file back into the central document. The previously downloaded file and updated file will then be deleted for security purposes.

1.3 Relation to Existing System

The Oracle Docs system that is in use by the CNS department at Virginia Tech suffers from multiple standards to look-up information about the inventory currently in the department's possession. This dated technology and with the vast availability of mobile products has led to the chance for the department to upgrade how the inventory is tracked. 21st-Inventory will not take the place of Oracle Docs, as the application will still need to access the central database provided by Oracle Docs; however, 21st-Inventory will be a proof-of-concept application that hopefully will pave the road for the shift away from Oracle Docs.

1.4 Expected Impact of the Project

The time lost in NI&S inventory tracking and reporting is staggering. With this app, we could quickly dispatch this task, and probably extend the functionality to find inventory on trucks and warehouse shelves.

1.5 Stakeholders

The main stakeholder is the client, NI&S, whom for the app is being developed. The three students, Brandon Dean, Elliot Garner, and Brannon Mason, also have a stake in that their semester project grades are dependent upon the delivery of the stated requirements. Lastly, Dr. Edward Fox is a stakeholder as he represents the students who are developing this app.

1.6 Roles in Project

Brandon Dean and Elliot Garner are the designated main programmers. As such, they will implement programmatically the desired design once in the prototyping phase of the project. Brannon Mason is the designated designer, a role in which he will be primarily responsible for the design of the app. This comes with the responsibility of creating sketches, wireframe diagrams, and use/case descriptions.

1.7 Timeline and Milestones

March 5: App GitHub repo created.
March 19: We want testing database to be sanitised and set up for use. We also will have a preliminary version of a barcode scanner implemented.
April 2: The app will be able to read a barcode and get its associated product data. We want to also manually update product info and add new items.

April 16: Finalized all barcode technology used, all information stored into a .csv file, and prototype running on a test device.
April 30: All deliverables completed

2. Requirements Specification

2.1 Specific Requirements

The following list contains the planned features that the application will handle:

- Barcode scanning to update product information
- Manual entry to update product information
- Asset-ID look-up
- Items loaned to a Custodian-ID look-up
- A central Google Document in .csv format
- Revision History
- Adding new items into the central inventory catalogue
- A security system to block unauthorized users
- The following fields of information to be stored:
    - Asset-ID
    - Custodian-ID
    - Description
    - Last known GPS Location
    - MAC Address

3. Architecture Specification

The 21st Century app will be developed in Swift, using XCode, targeted for devices running iOS 8. Our test emulator target is the iPhone 6 Plus, but can also be run on other devices running iOS 8, including the iPhone 4s, 5, 5s, and iPhone 6. In addition, any iPad which can run iOS 8 can also run this app. So this user pool also includes the iPad 2, iPad 3, iPad 4, iPad Air and iPad Air 2. Every effort will be taken to use plug ins that are written in Swift and use free software to reduce costs and complexity.

4. Design Specification

4.1 Wireframes

In the preliminary design, the 21st Inventory will support 3 different methods to manipulate the inventory database. This includes a method to add inventory objects, a method to update inventory objects, and a method to get information on inventory objects. For more information on the design and flow of the current user interface see the wireframe in **Appendix A**.

4.2 Data Flow

When the user opens the first thing the application will present to them will be presented with a camera view to take a picture of the barcode with. The user will then line up the barcode inside of a predefined box, so that the application can recognize the barcode properly, and will take a picture of it.Once the user has taken a picture of the barcode the application will scan the picture, read the barcode, and get the information stored in that barcode. Then we will download the information stored in a google spreadsheet and transform it into a table. Our application will then search through that table for the barcode data.

If the barcode lookup was successful and the data for the barcode was found then the user will be presented with the data associated with the barcode. Then the user will be have the option to edit the information. If they do the the edited information will be pushed back to the spreadsheet and overwritten.

If the barcode lookup was unsuccessful and the data was not found then the user will be presented with the option to create data associated with that barcode. When the user has filled out all of the information available to them then that new information will be written to the google spreadsheet.

5. Implementation Specification

5.1 Implementation Summary

Our project will be an iOS application written for the following apple devices: iPhone 4s, iPhone 5, iPhone 5s, iPhone 6, and iPhone 6 +., as well as any iPads newer than iPad 2. It

will be developed in XCode using the Swift programming language. We will be using the Google Docs API,

5.2 Project Timeline/ Milestones

As covered in Section 1.7, we have identified milestones at approximately 2 week intervals, during which project stakeholders can evaluate project progress with the team.

March 5: App GitHub repo created.
April 2: We want testing database to be sanitised and set up for use. We also will have a preliminary version of a barcode scanner implemented.
April 9: The app will be able to read a barcode and get its associated product data. We want to also manually update product info and add new items.

For graphical layout on the timeline see **Appendix B**

5.3 Technologies

As specified in the Architecture Specification (**Section 3**), we plan to develop our app in XCode, using the Swift programming language for iPhone devices. We will have to identify and implement an external plugin for handling the capture of barcodes and exportation of relevant data to a .CSV file. The testing database will be hosted in Google Docs, using relevant APIs.

The project repository is privately hosted on Virginia Tech's GitLab site. This can be found at git@git.cs.vt.edu:deanbr/21stinventory.git. This has already been created, and has all three team members listed as collaborators. The advantage for this setup is that it will be easy to share code with all stakeholders, while being able to migrate the project at any time if such an action is desired.

5.4 Implementation Steps

The actual application will take place in a set of seven steps as follows:

1. The given data and information will be checked whether or not it currently is a parseable .csv file. If not, the data will be scripted to create a .csv file that will be hosted on Google Drive.

2. To easily update and search the .csv file, the information that will be entered into the .csv file will be sorted by barcode number (ideally the ID number that is part of every device).

3. With the parse-able files set up in their correct form, the next portion will be the implementation of generic data structures to hold the information that will be parsed in from the .csv file. The data structure to hold the information will most likely be a combination of a Hash Table and a balanced binary search tree, which will be an AVL tree or a Fusion tree.

4. Once the data structures have been tested and finished, the look-up functions will be derivative methods that pull from the data structure methods. We will be checking the barcode against a server of data on the back end of the project, supplied by the client.
5. The barcode scanner will set-up to specifically test that the information from the barcode can be supplied to the model portion of the application
6. The view portion and the controller portion of the application will be finally synced together. The lookup functions will be attached to specific widgets on the GUI pages created.
7. Finishing touches and testing the full functionality will be the final step to the implementation of the application

5.5 Documentation

Throughout the project, any code that is written will be commented thoroughly, with efforts taken to ensure ease of understanding for developers working with our code base after this semester. After the project, a short “Developer’s Manual” will be published as part of the final report which will contain our experience with developing the project, how it is structured, and how everything is implemented.

6. Prototyping

6.1 Process

We have not yet made any prototypes, but have discussed what the prototype shall look like. It will be done in Swift and made with the XCode IDE so that commonality with the final product is easily achieved.

We will start with programming front end interface first, such that the user will be able to navigate (i.e. click around) the app, but it will not actually do anything behind the scenes (i.e. connect to a server, send results after scanning, etc.).

Following that step, we will behind incrementally adding features, in the order of complexity. So we will tackle a small problem, finish it, and move to the next. We will continue this iterative process until our final product is ready to be delivered.

6.2 Implementation of Prototype

As covered above in 6.1, we will implement this in the language and development environment expected of the final product. Features will be expected to be minimal at first, but incrementally added as time approaches the final deadline (optimally also in line with our projected milestones).

6.3 Expected Changes Planned

Right now, we have mockups of the system given to us by Kimberley (our main stakeholder). These can be found in **Appendix C** (the Excel spreadsheets with data provided). We are going to take this data, in the form of a .csv file, and send this to the server currently handling the current implementation.

Further, due to the design of the files given to us by the main stakeholder, we will try to further collaborate all the information that has been given to us to possibly re-design how the information is presented. In its current form, most of the information is scattered across over 14 files. This needs to be fixed so that we can move on from here. Will we be meeting with our stakeholder on the week of 3/29 to possibly update and secure the correct design for our prototype going forward.

6.4 Refinement Stage

At the current time, the project has been updated from storing multiple different types of inventory items to just one type. The 21st Inventory project will now be used to help the implementation of installing multiple wireless access points (WAPs) throughout the dorms and buildings on the Virginia Tech campus. Specifically, the application will track where the WAPs will be placed, from the warehouse to the room, as well as holding the information on the MAC addresses on the inventory numbers.

The reason for this update is to make sure there is no loss of materials when the installation of access points begins, as well as making sure that all the items are installed in the correct locations for the quality assurance team to double check quickly and effortlessly. Further, the .csv file created will help the CNS department easily tell the finance committee which buildings they will have to bill and how much each building will have to be billed. The information will be stored in a similar file as shown in **Appendix D**.

As it currently stands, the application will take, as input, a photo of a barcode. The barcode will be of either the the serial number, the CNS, or the P-tag, the last two are added onto the back of the WAP. Upon scanning, if the scanned information cannot be found, then it will be added to the database, along with the rest of the required barcodes, the MAC address of the device, and the location of the user. When the user scans any of the four barcodes and the information is found, it will update the location of the corresponding item, and allow the user to edit the current information to the item if it was inputted incorrectly.

As of the April 8, 2015, the projected number of items to be scanned is in the vicinity of 3000 I devices. There are going to be roughly 5 or 6 corresponding pieces of data attached to each device (i.e. MAC address, text location, gps location, serial number, CNS number, ID number, and ptag). The last few pieces of data being tracked were added in as of that last meeting, on April 3, 2015. In addition, it was requested we add in another screen so the app could also store whether the device is hanging on the wall or ceiling. And while it isn't very intensive necessarily to store a few additional pieces of data, this change of scope has also changed how we need to design our app and has sent us back to the drawing board to rework our approach. As such, the (demo-able) prototype is on track for an April 13th debut in our meeting with our client.

Some items we need to keep in mind going forward are whether there will be different barcode types, how we will need to treat retrieving gps data within buildings here at Virginia Tech, where wireless transmissions have trouble indoors, and how to integrate our data with a sample database. We are hoping making a working prototype will help answer the questions and give the client a chance to test out their product.

With the new update of the project spec we have updated the wiki to accurately reflect the current project.

6.5 Implementation Steps Update

With the new specifications wanted by the client, the implementation of the updated application will take place in a set of 6 steps as follows:

1. The design for the model of the application will be created following the given data elements that the client has provided. This includes, but is not limited to, writing up the basic functions that will be a wrapper to be called by the controller portion of the application and the making of the parsing unit for the file to be updated.

2. Resolving final bug testing with barcode scanning and photo recognition. This includes acquiring an Apple product for every member of the team for reliable testing. These products will be rented from Innovation Space.

3. To overcome the slowness of multiple I/O calls to update each line, the application will store the file a second time over as strings in a hash table. This allows for a quick look-up time, as well as, only forcing the file to actually be updated upon closing of the application.

4. The barcode scanner will set-up to specifically test that the information from the barcode can be supplied to the model portion of the application.

5. The view portion and the controller portion of the application will be finally synced together. The lookup functions will be attached to specific widgets on the GUI pages created.

6. Finishing touches and testing the full functionality will be the final step to the implementation of the application.

6.6 Status Since Refinement I

Our team met with Kimberley on Monday April 13, 2015 to show the prototype and discuss progress. Here, we clarified what data she is expecting from the app, and what she should expect as the deadline approaches. We agreed to two more meetings before the deadline, with the next one being Wednesday, April 22 and the next a week after. At this first meeting, we will have everything in the app working, with the exception of it communicating to the database. At the latter date, the database functionality will hopefully be implemented, and, if not, it will export to .CSV (as was originally planned and expected). See **Appendix F** for photos of the views of the prototype shown to Kimberley in the meeting.

7. Testing

7.1 Changes Since Last Revision

Since the last report, the system has been updated to download files from the drive and to correctly parse the files into the application. Further, the authorization screen has been finished, as well as, the ability to jump from window to window as envisioned. Finally, the barcode scanner works and correctly scan all three possible barcode IDs that the client had required to be produced. In addition, we have expanded the scope of the app to include a main splash screen, which the user will first see upon loading the app. From here, the user can select to use the camera, or navigate to the settings screen. If the user selects the camera, they then proceed along the same logical flow described in the previous report.

7.2 Prototype

The prototype has been further updated to include the full scope of the project. At this point, the prototype exists to show the full functionality of the final product. The next step is for our group to hold our scheduled meeting on April 28, 2015 to combine the two parts of the project: The app interface with which the user interacts, scans a barcode, and enters any data. And the code interacting with Google's Spreadsheet API which will allow us to take any data entered by the user in the app and push it to a database hosted in a Google Spreadsheet.

In addition, we can also plan to have our app output a .CSV file for the client to manage at their convenience, as was the original plan. However, our code working with Google's Spreadsheet API supercedes this functionality and makes any changes directly to a database owned and maintained by NIS.

This latest prototype, which should have the "final product" designation applied to it following a successful meeting with Kimberley Homer on April 29, 2015 during which we will sign off on deliverables met and hopefully leaves with a "final," or "near-final" product to which we will make the necessary changes.

7.3 Problems Encountered

Multiple problems and bugs have been encountered with our system, including, but not limited to, the following:

- If barcodes are too close, the scanner is unable to correctly read the barcodes. This can be remedied by either placing the barcode strips on the inventory item further apart or by covering adjacent barcodes so the scanner only has a single code to look at.
- Due to the lack of an actual API for Google's Drive, there was a major difficulty in trying to even get the files from the Google Drive. At this time, we have successfully gotten around trying to constantly look up the address to the file, and rather are using the metadata from the file.
- There were a couple of minor bugs with actually parsing the file correctly and making sure that the application wrote to the right location, but these were easily resolved when debugged.

8 User's Manual

8.1 Contents

The following is a list of files that have been included in the ScannerTest.zip file with our submission to VTechWorks and a description of each:

Scanner Test - This folder contains all of the assets for our project. Source code, images, files to write to while the application is running. Look below to find out how to properly view the contents of this folder

Scanner Test.xcodeproj - This folder is how you properly view the project using XCode on a Mac. By using XCode to open this folder you can edit all aspects of the project. It is strongly recommended that if you alter the project in any way you do it through XCode.

Scanner TestTests - This folder is meant to host all of the test files for the project. For our purposes of the project we didn't write any tests for the project, so the folder is mostly empty. But this is where test files were to go if you were to write them on a future date

google-api-objectivec-client-read-only - This folder is a folder downloaded from the Google Drive API. It contains all of the files required for getting an application integrated with any Google Service. **WARNING: Do not move the location of the project on a computer without resetting the "User headers search path" in build settings to the path of the Source folder within his folder. Please make sure that the path to the source folder has no spaces in it either.**

8.2 Manual

The 21st-Inventory application was developed to mobilize and centralize the tracking of the inventory of the Network Infrastructure & Services (NI&S) while they installed new equipment inside Virginia Tech buildings.

In order to use the application the user must have access to the "arubawaps.csv" file on Google drive. If the user does not have access to the file, access can be granted to them by sharing the file with them through the Google drive.

Upon opening the application, if the user has not already done so, they should wait until the application presents them with a Google account login screen. This screen will ask the user to sign into a Google account and can be seen in **Figure 8.2.1**. For ease of access the user should sign into the approved NI&S account that already has access to the appropriate file on the drive. For the username and the password to this account please email Kimberley Homer at homerk@gmail.com.

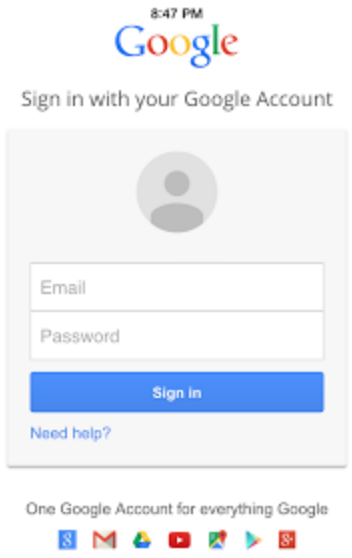


Figure 8.2.1. The layout of this screen is similar to any standard Google Login screen, as this is trying to access the User's Google Drive

Once the user has signed in they should press the camera button on the main screen to scan barcodes. This will take the user to the following screen, where they will then be allowed to scan barcodes. At the bottom of that screen are buttons "VT ID", "C Number", "Serial Number", and "MAC Address". When the user scans a barcode with the respective button selected it will store that information under the respective field.

In order to scan a barcode the user must have the barcode in the center of the camera display, allow it to correctly adjust to the distance from the barcode, and wait the for green box to appear. Once the green box has appeared it is then appropriate to scan another barcode. After the user has scanned all barcodes needed, then press the "Proceed" button to move onto the next screen.

This screen has all of the information that the user can input. If the user scanned barcodes while the buttons "VT ID", "C Number", "Serial Number", and "MAC Address" that information will be presented to the user. They must then input the current location of the device, the stock number, current custodian, and room number of the device's current status. If any of these fields do not apply, please leave them blank. The layout of the screen and what can be found on it is presented in **Figure 8.2.2**.

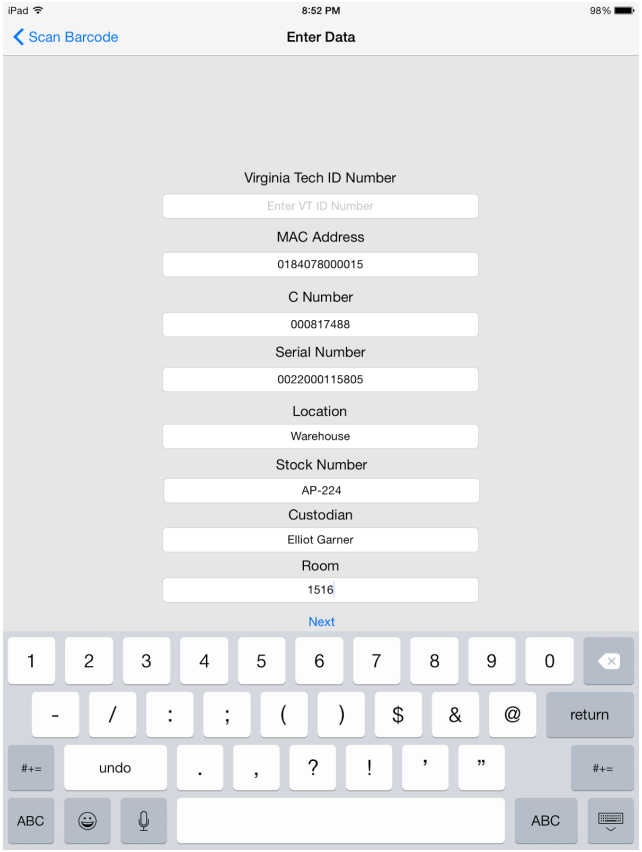


Figure 8.2.2. An example of the filled in fields with information that has been manually entered or scanned in

When clicking next the user will be allowed to select whether or not the device is wall mounted or ceiling mounted or will be able to enter a custom location of the device by hand.

When hitting next another time the user will be presented with all data that the user has entered so far. If any of this data is incorrect, they will be able to go back to the appropriate screen and correct it. If all of the information is correct the user should hit the next button. Then select the Submit button. Upon doing that the updated information will be uploaded to the appropriate file on the Google Drive. All of this can be found in **Figure 8.2.3**. If the user as anymore issues understanding how the flow of the application works, they can consult the wireframe in **Appendix F**.

Figure 8.2.3. The final information review screen that allows the user to check over what currently resides in the fields and allows them to either move forward or change values

9. Developer's Manual

9.1 Front End

For our project we have two variables inside of our AppDelegate that we use as global variables throughout the entire application so we don't have several instantiations running around at the same time. First we have the fileParser variable, a FileParser21 object, which holds a copy of all of the data read from the Google Drive File, so it's important to not have more than a single instance of that variable. Next we have the scanner variable, a DriveScanner object, which is the item that you call all of the Google Drive operations through and holds your validation information in it. Again, it's important to not have more than one copy of that variable.

When the user first opens the application they will be presented with the main screen. Inside of the ViewDidAppear method we check a flag to see if we have asked the user to sign into Google Drive yet. If they have, we move straight to retrieving the file contents from the file hosted in the Drive. If we haven't, then we need to set up the GTMOAuth2ViewControllerTouch View Controller shown in **Figure 8.2.1**. This process is described in depth in **Section 9.2**. Once the user has logged in, they are then free to press the Main Screen button to segue into the camera view.

The camera view, where the user actually scans the barcode, uses a lot of code inspired by code at The Bowst Blog (Bowst). The few major differences are that we added a segmented control to the bottom. This allows the user to select which barcode they are scanning, and stores it in memory based on what portion of the segmented control was selected when the barcode was scanned. When the user scans an object, we do a look up in the dictionaries stored in memory, described in more detail in **Section 9.2**. If that data already exists we present it to the user in the next screen immediately. If it's not found, this data is stored into the variables strVTIDNumber, strMacAddress, strCNumber, and strSerialNumber. Then when the user presses the next button, a segue is performed passing those variables to the next screen, the DataEntryViewController.

This screen is just a list of labels and text fields. The application populates the VTID, the Mac Address, the C Number, and the Serial Number, based on whether or not the barcode was scanned on the previous screen. If it was, then the data was passed to this screen and automatically set as the values of those text fields, else they will be blank. The user is then able to edit any of the other text fields as they see fit. We do not do any validation on any of the user input. They are able to input any combination of characters together that they want.

Once the user has finished inputting the data and have pressed next, all of that data is passed to the next screen, WallMountViewController. This screen has two buttons and a text field. One button for wall mounted, one for ceiling mounted, and the text field is so that they can enter their own location of the device. If they pressed either of the buttons that passes to the next screen, in addition to all of the previous data, the model number of the device. If they entered a custom value that value will be passed as the part number.

Next all of the data that has been passed will be presented to the users in the form of labels. These are non-editable and the values of the labels are populated with the respective data that has been passed forward. The alignment on these labels can be kinda wonkey and will not always look perfect.

Once the user has pressed next that specific record has been recorded into the files memory. If they hit the submit button on the next screen, the current list of records is sent to be written to the Google Drive. The user is then brought back to the main screen.

9.2 Drive Integration

The Drive Integration is mainly contained within three different classes, the DriveScanner class, the Record class, and the FileParser21 class. The Record class is used to create record objects to store all of the value pertaining to a single device. You create a new record by passing it all of the relevante information that you have pertaining to the device. Inventory number, serial number, mac address, stock number, building id, building name, building abbreviation, room number, model number, and description. You can turn a record object into a string by taking each portion of the record, adding a comma do it, and concatenating them. This is done because we are writing them to a csv file, so we need to separate fields by commas.

The FileParser21 class has 3 main functions. First is addToHash. This reads in all of the data written to a file within memory and, because the file on the Drive is a csv file, seperates them by commas, and then creates a Record out of those and stores those into a dictionary where the key is the inventory number and the value is the record object. We also have two other dictionaries. One connection the serial number to the inventory number, and one connecting the mac address to the inventory number. This is so that if the users scans another barcode other than the inventory number, we can look up if the item exists in the dictionary for them to edit. Next is the writeToFile function. This function takes every object stored in the primary dictionary, convert the records to strings, and then write them to the file in memory. Finally you can add an item to the dictionary by passing it all of the relevant information described previously.

Third we have the DriveScanner class. This class is where all of the drive integration comes into the project. First we have the userLogin function. Each project registered with Google has a client ID and a secret. The userLogin function sends those to google for authorization and returns and authorizer if those are correct. We store that authorizer inside of the global drive scanner. We then call createAuthController to create the login screen to present to the user. After storing that inside of a variable we return it to the main screen to be presented.

The createAuthController function creates and returns a new GTMOAuth2ViewControllerTouch screen with the client secret and client id. However, once the view controller has been created it calls viewController when it finished through a selector. The viewController function sets all relevant information for authorization on completion. If there was an error creating the authController we present it to the user and do nothing else. If there wasn't, we set the error to the authentication result, set the access token to the authentication result token, then dismiss the view controller, and finally retrieve the file contents through the retrieveFileContents function.

The retrieveFileContents function is where the data is retrieved from the file in the Drive. To do this you create a GTLQueryDrive with the query that you want. We create a queryForFilesList query, which returns all of the files that the user has access to. we use the

driveService to execute that query with a completion handler. On completion if there was no error we need to iterate through the GTLDriveFileList that the query returns. We then check each file's originalFileName against the file that we're looking for. Once we've found the file we need to retrieve the file's downloadUrl, the url that actually downloads the file's data. we create a NSURL request using the download url and then convert that NSURL request into a NSMutableNSURL request. We do this because we need to attach a custom header to each request to google drive in the following form "Authorization: Bearer [accessToken]." We set the value of the mutable request and open a new NSURLConnection sending an asynchronous request with a completion handler. If that request returns an error then we don't have proper validation and we need to create a new auth controller, which recalls retrieveFileContents on completion. If there is no error then we read the data from the file, write that data to the file i memory, and then add the data in the file to the dictionaries using the addToHash function.

WARNING: Because all of this is done during the viewDidAppear method there there is the possibility of the user to try to interact with the main screen while all of this is running. This leads to the possibility of them interacting with the screen while file contents are being retrieved. This can lead to unexpected results. An appropriate way to fix this in the future would be to show a loading view while the contents of the file are being read. One the contents are downloaded, you could remove the view and let the user interact with the application again.

The last important method in the DriveScanner is the uploadFile function. This creates a new query, a queryForFilesUpdateWithObject query, which is used to overwrite a file already existing on the drive. We attach the file in memory to that query. We then execute that query with the driveScanner.

The drive scanner interacts with the application in two main places. First in the Main Screen view controller. The user login function, and by extension the retreiveFileContents function are all called in the viewDidAppear method. Finally, when the user returns to the Main Screen from the Finished Screen, it calls the uploadFile function in drive Scanner. The DriveScanner methods are very self contained and were made to interact with the user and the front end as little as possible. Any questions about the Drive API should be directed to Google's Website.

10. Final Notes

10.1 Lessons Learned

In the beginning, the client wanted an inventory system that would be able to store multiple different objects ranging from computers to telephones that could become mobile and allow for employees to reliably track inventory items. However, over time this had to change. For

over a month, the team had to remain in a delayed-development state, as information on what specifically was wanted, like the file set-up, continued to oscillate.

About a month before the project deadline, the product objective was finalized. Instead of tracking all inventory values, the team needed to create an application that only focused on storing Wireless Access Points (WAPs) information, as the NI&S department would be installing a large amount of them in the near-future.

What we learned from this was for us to need to have more communication with the client and have more communication with each other. Without good communication, we had difficulties understanding what we needed to do for a long period of time that could have been better spent creating a larger scale product, as well as, having difficulties on solidifying out design. However, this is an internal critique of what we could do better in the future.

10.2 Acknowledgements

We would like to thank both Kimberley Homer and Sarah Crowder, our clients at the NI&S, for their incredible help and flexibility when it comes to this project. They have been very understand and patient when it comes to the project and its specifications.

Next we would like to thank Dr. Coleman Fox for setting up original communications with Ms. Homer and Ms. Crowder and the project.

Finally we would like to thank Virginia Tech's Innovation Space for letting us rent iPads to do testing of our software on.

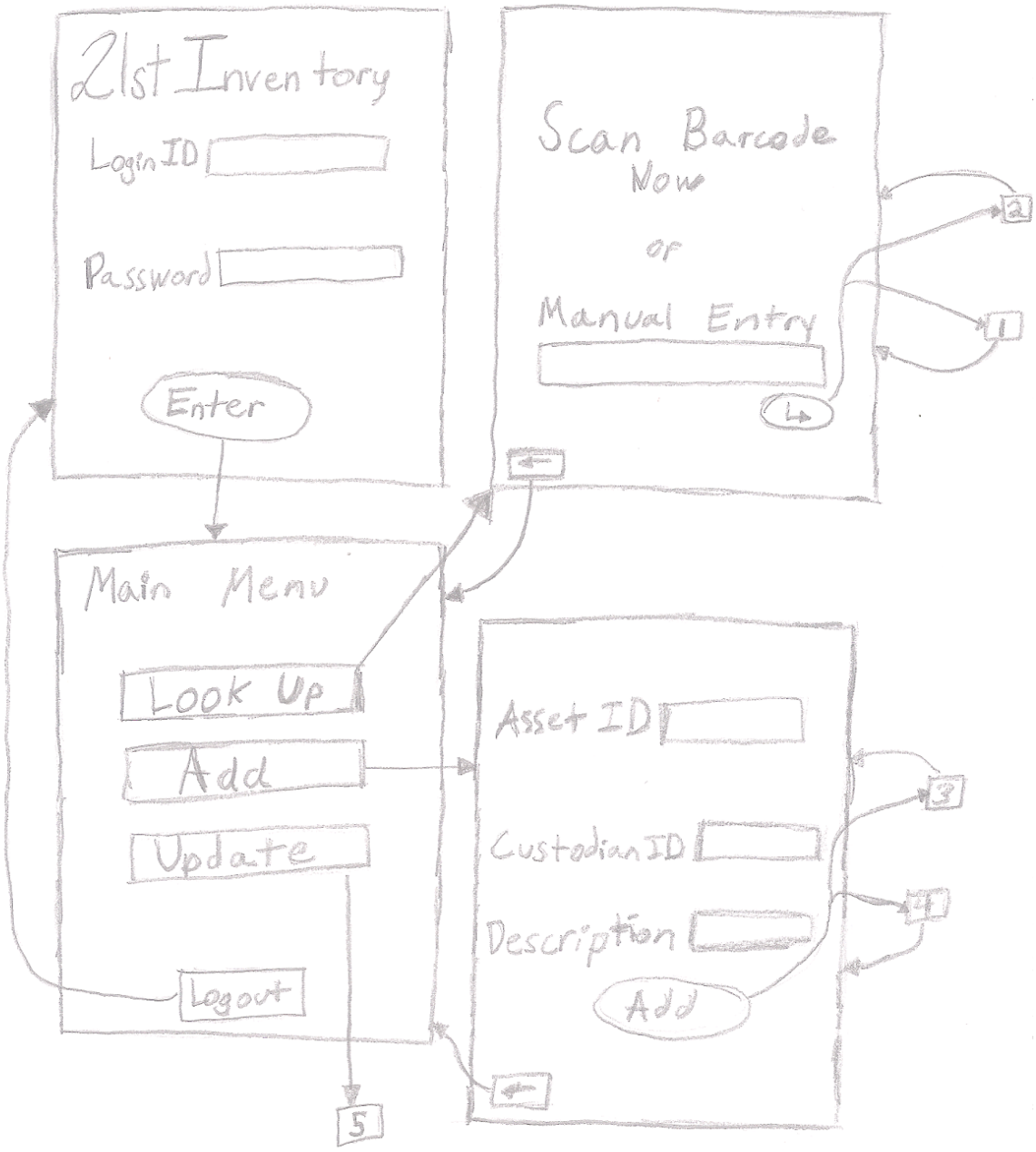
10.3 References

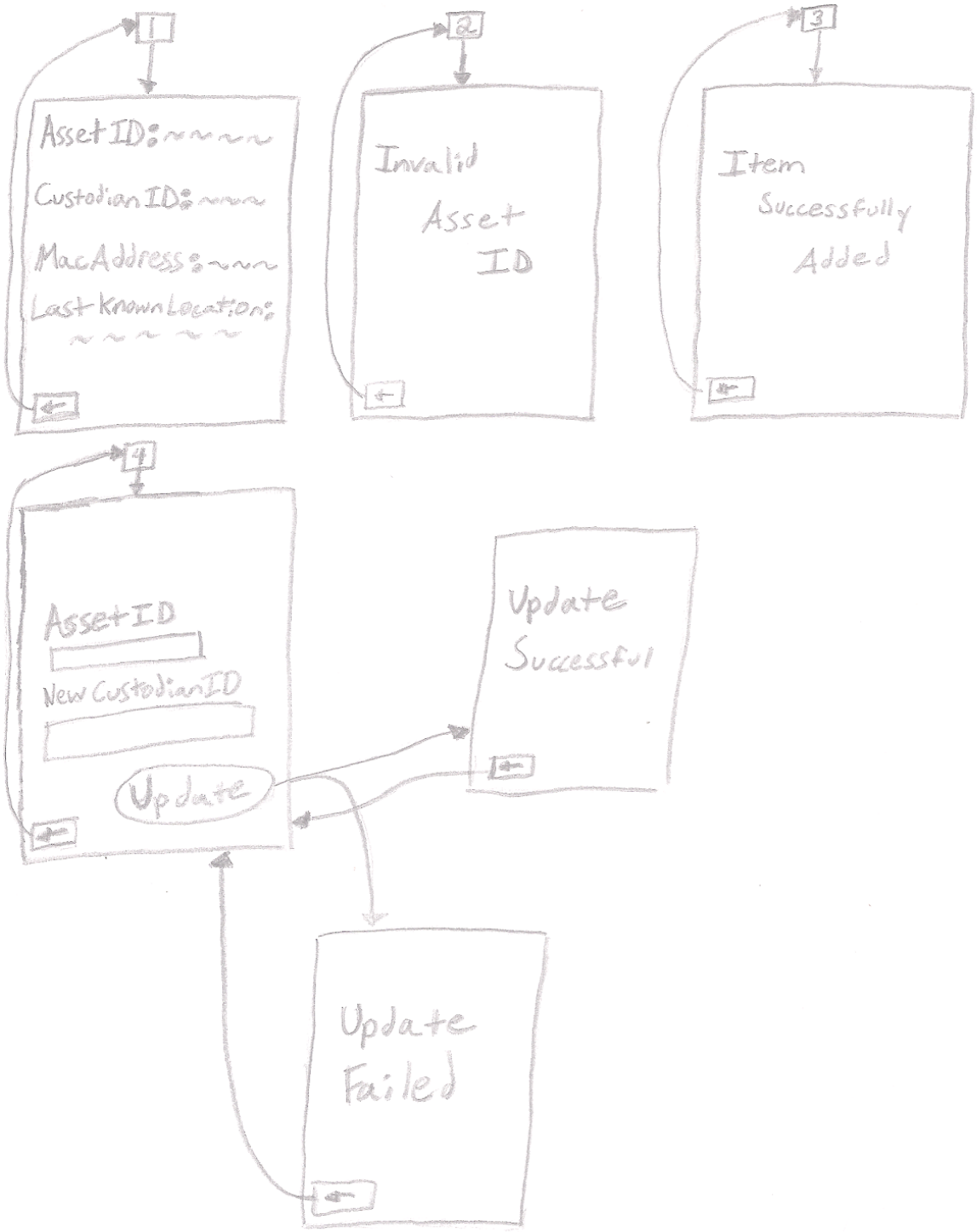
"The Bowst Blog." *Simple Barcode Scanning with Swift*. Bowst, n.d. Web. 30 Apr. 2015. <http://www.bowst.com/mobile/simple-barcode-scanning-with-swift/>.

"Introduction to the Google Drive Platform for IOS." *Google Developers*. Google, 20 Mar. 2015. Web. 30 Apr. 2015. <https://developers.google.com/drive/ios/intro>.

"The Swift Programming Language: About Swift." *The Swift Programming Language: About Swift*. Apple, n.d. Web. 30 Apr. 2015. <https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/>.

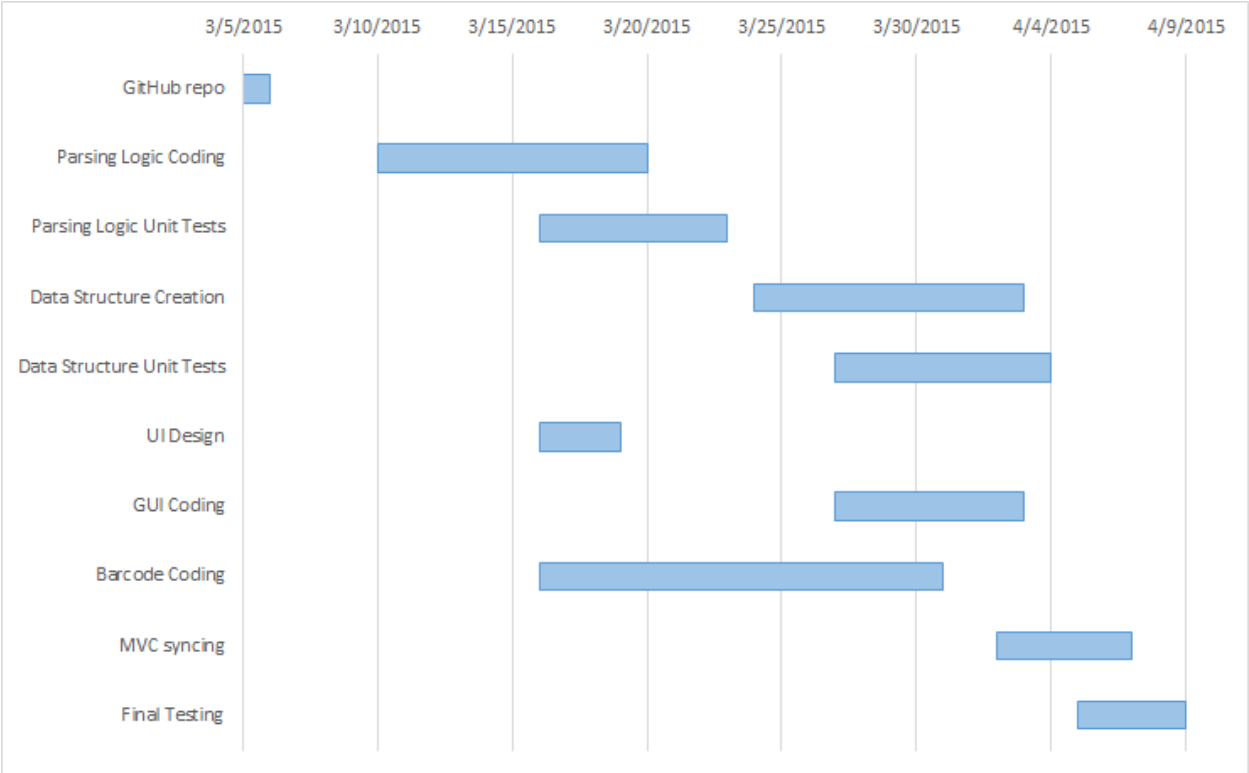
Appendix A: Flow Chart


21st Inventory
Login ID
Password
Enter
Scan Barcode
Now
or
Manual Entry
2
1
4
Main Menu
Look Up
Add
Update
Logout
5
Asset ID
Custodian ID
Description
Add
3

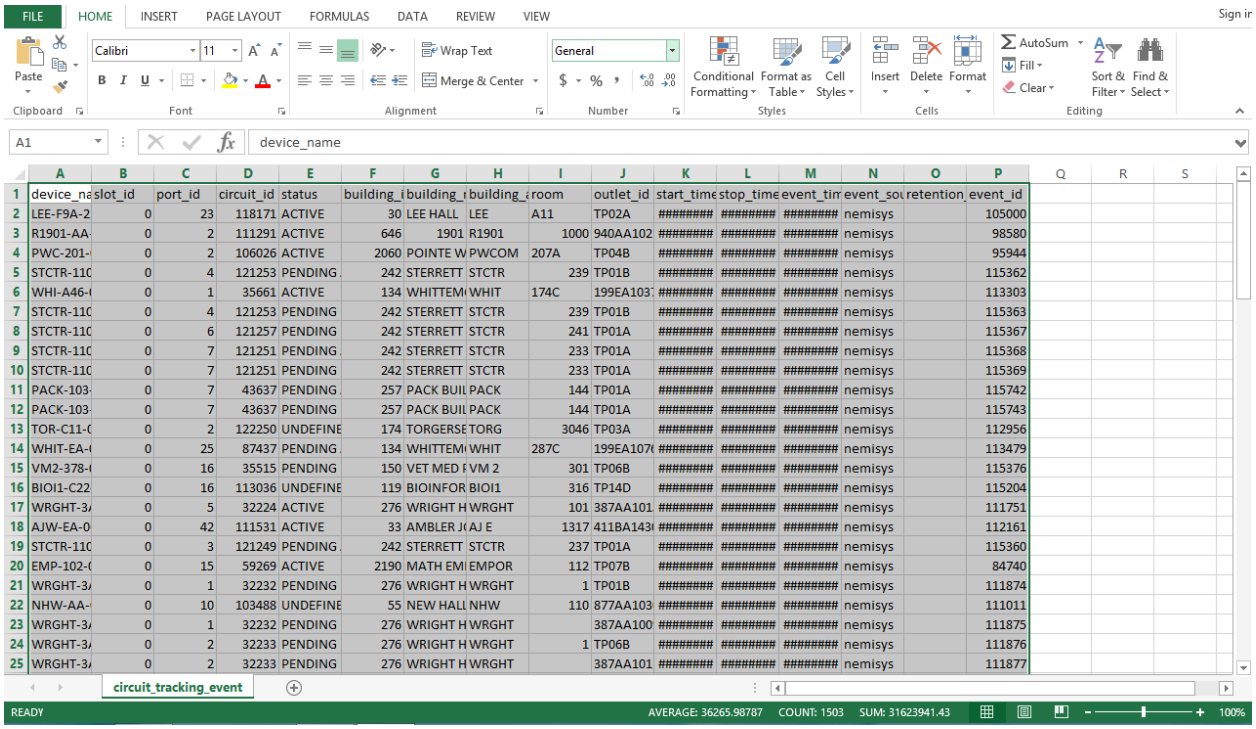

1
Asset ID:
Custodian ID:
Mac Address:
Last known Location:
2
Invalid
Asset
ID
3
Item
Successfully
Added
4
Asset ID
New Custodian ID
Update
Update
Successful
Update
Failed

Appendix B: Work Schedule

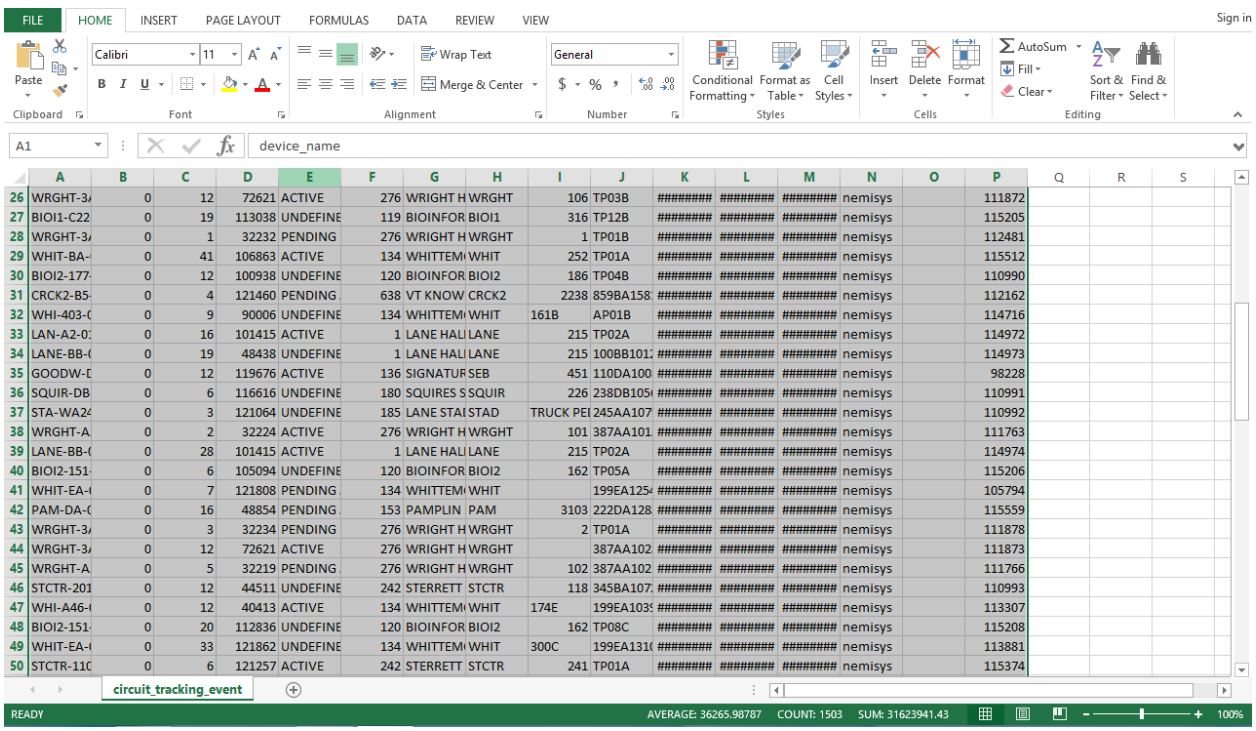
Gantt Chart



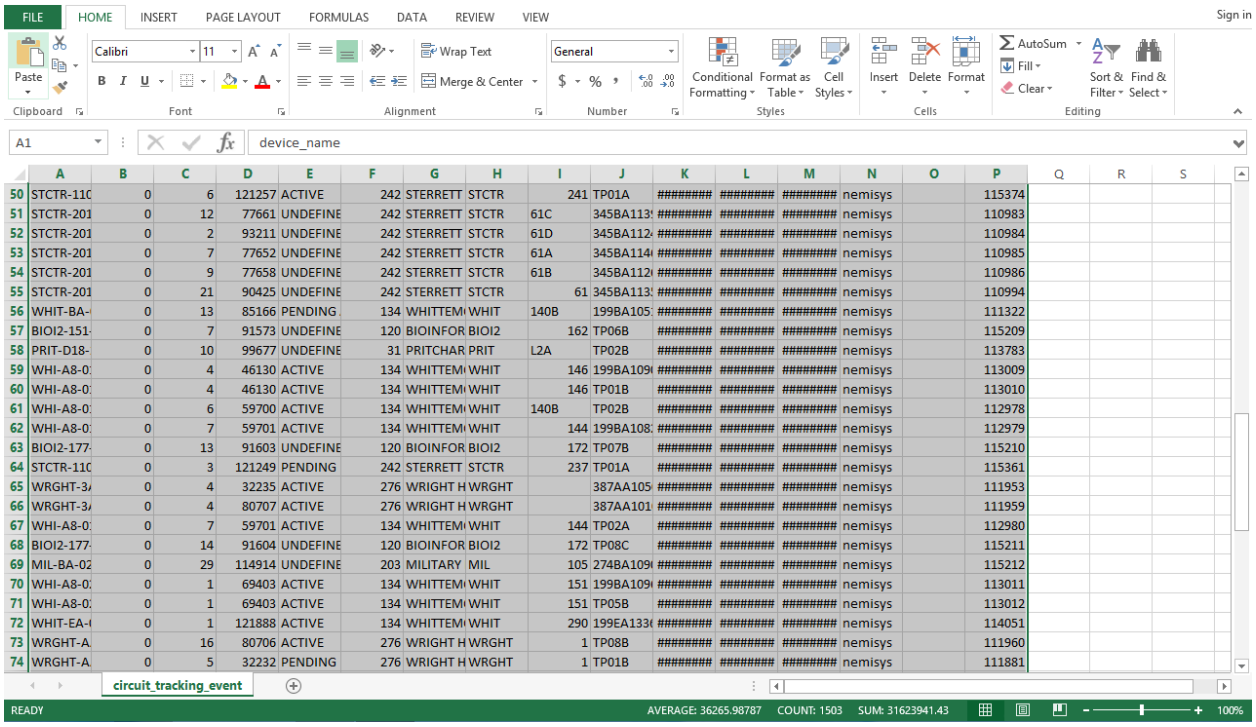
Appendix C: Sample Data



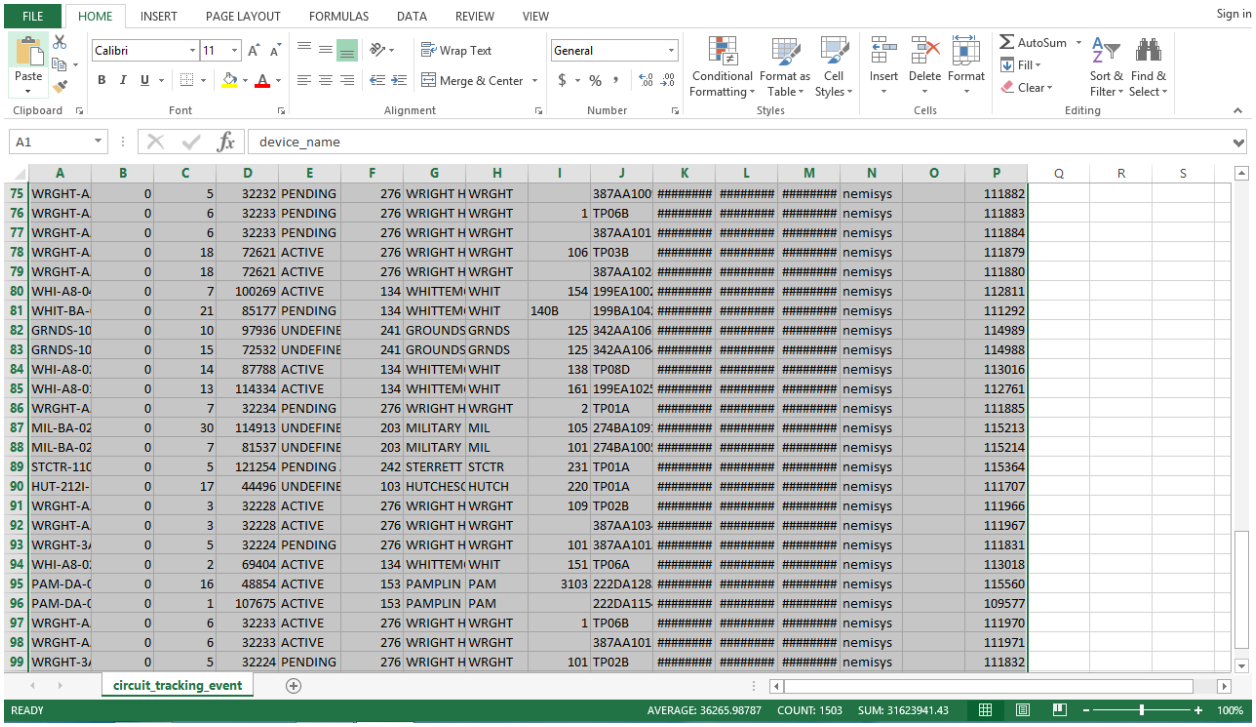
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	device_na	slot_id	port_id	circuit_id	status	building_i	building_	building_	room	outlet_id	start_time	stop_time	event_tin	event_so	retention	event_id
2	LEE-F9A-2	0	23	118171	ACTIVE	30	LEE HALL	LEE	A11	TP02A	########	########	########	nemisys		105000
3	R1901-AA-	0	2	111291	ACTIVE	646	1901	R1901	1000	940AA102	########	########	########	nemisys		98580
4	PWC-201-	0	2	106026	ACTIVE	2060	POINTE W	PWCOM	207A	TP04B	########	########	########	nemisys		95944
5	STCTR-11C	0	4	121253	PENDING	242	STERRETT	STCTR	239	TP01B	########	########	########	nemisys		115362
6	WHI-A46-(	0	1	35661	ACTIVE	134	WHITTEM(	WHIT	174C	199EA103	########	########	########	nemisys		113303
7	STCTR-11C	0	4	121253	PENDING	242	STERRETT	STCTR	239	TP01B	########	########	########	nemisys		115363
8	STCTR-11C	0	6	121257	PENDING	242	STERRETT	STCTR	241	TP01A	########	########	########	nemisys		115367
9	STCTR-11C	0	7	121251	PENDING	242	STERRETT	STCTR	233	TP01A	########	########	########	nemisys		115368
10	STCTR-11C	0	7	121251	PENDING	242	STERRETT	STCTR	233	TP01A	########	########	########	nemisys		115369
11	PACK-103-	0	7	43637	PENDING	257	PACK BUIL	PACK	144	TP01A	########	########	########	nemisys		115742
12	PACK-103-	0	7	43637	PENDING	257	PACK BUIL	PACK	144	TP01A	########	########	########	nemisys		115743
13	TOR-C11-(	0	2	122250	UNDEFINE	174	TORGERSE	TORG	3046	TP03A	########	########	########	nemisys		112956
14	WHIT-EA-(	0	25	87437	PENDING	134	WHITTEM(	WHIT	287C	199EA107	########	########	########	nemisys		113479
15	VM2-378-(	0	16	35515	PENDING	150	VET MED I	VM 2	301	TP06B	########	########	########	nemisys		115376
16	BIOI1-C22	0	16	113036	UNDEFINE	119	BIOINFOR	BIOI1	316	TP14D	########	########	########	nemisys		115204
17	WRGHT-3A	0	5	32224	ACTIVE	276	WRIGHT H	WRGHT	101	387AA101	########	########	########	nemisys		111751
18	AJW-EA-0	0	42	111531	ACTIVE	33	AMBLER J(	AJ E	1317	411BA143	########	########	########	nemisys		112161
19	STCTR-11C	0	3	121249	PENDING	242	STERRETT	STCTR	237	TP01A	########	########	########	nemisys		115360
20	EMP-102-(	0	15	59269	ACTIVE	2190	MATH EMI	EMPOR	112	TP07B	########	########	########	nemisys		84740
21	WRGHT-3A	0	1	32232	PENDING	276	WRIGHT H	WRGHT	1	TP01B	########	########	########	nemisys		111874
22	NHW-AA-(	0	10	103488	UNDEFINE	55	NEW HALL	NHW	110	877AA103	########	########	########	nemisys		111011
23	WRGHT-3A	0	1	32232	PENDING	276	WRIGHT H	WRGHT		387AA100	########	########	########	nemisys		111875
24	WRGHT-3A	0	2	32233	PENDING	276	WRIGHT H	WRGHT	1	TP06B	########	########	########	nemisys		111876
25	WRGHT-3A	0	2	32233	PENDING	276	WRIGHT H	WRGHT		387AA101	########	########	########	nemisys		111877



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
26	WRGHT-3A	0	12	72621	ACTIVE	276	WRIGHT H	WRGHT	106	TP03B	########	########	########	nemisys		111872
27	BIOI1-C22	0	19	113038	UNDEFINE	119	BIOINFOR	BIOI1	316	TP12B	########	########	########	nemisys		115205
28	WRGHT-3A	0	1	32232	PENDING	276	WRIGHT H	WRGHT	1	TP01B	########	########	########	nemisys		112481
29	WHIT-BA-(	0	41	106863	ACTIVE	134	WHITTEM(	WHIT	252	TP01A	########	########	########	nemisys		115512
30	BIOI2-177-	0	12	100938	UNDEFINE	120	BIOINFOR	BIOI2	186	TP04B	########	########	########	nemisys		110990
31	CRCK2-B5-	0	4	121460	PENDING	638	VT KNOW	CRCK2	2238	859BA158	########	########	########	nemisys		112162
32	WHI-403-(	0	9	90006	UNDEFINE	134	WHITTEM(	WHIT	161B	AP01B	########	########	########	nemisys		114716
33	LAN-A2-0:	0	16	101415	ACTIVE	1	LANE HALI	LANE	215	TP02A	########	########	########	nemisys		114972
34	LANE-BB-(	0	19	48438	UNDEFINE	1	LANE HALI	LANE	215	100BB101	########	########	########	nemisys		114973
35	GOODW-D	0	12	119676	ACTIVE	136	SIGNATUR	SEB	451	110DA100	########	########	########	nemisys		98228
36	SQUIR-DB	0	6	116616	UNDEFINE	180	SQUIRES S	SQUIR	226	238DB105	########	########	########	nemisys		110991
37	STA-WA24	0	3	121064	UNDEFINE	185	LANE STAI	STAD	TRUCK PE	245AA107	########	########	########	nemisys		110992
38	WRGHT-A	0	2	32224	ACTIVE	276	WRIGHT H	WRGHT	101	387AA101	########	########	########	nemisys		111763
39	LANE-BB-(	0	28	101415	ACTIVE	1	LANE HALI	LANE	215	TP02A	########	########	########	nemisys		114974
40	BIOI2-151-	0	6	105094	UNDEFINE	120	BIOINFOR	BIOI2	162	TP05A	########	########	########	nemisys		115206
41	WHIT-EA-(	0	7	121808	PENDING	134	WHITTEM(	WHIT		199EA125	########	########	########	nemisys		105794
42	PAM-DA-(	0	16	48854	PENDING	153	PAMPLIN	PAM	3103	222DA128	########	########	########	nemisys		115559
43	WRGHT-3A	0	3	32234	PENDING	276	WRIGHT H	WRGHT	2	TP01A	########	########	########	nemisys		111878
44	WRGHT-3A	0	12	72621	ACTIVE	276	WRIGHT H	WRGHT		387AA102	########	########	########	nemisys		111873
45	WRGHT-A	0	5	32219	PENDING	276	WRIGHT H	WRGHT	102	387AA102	########	########	########	nemisys		111766
46	STCTR-201	0	12	44511	UNDEFINE	242	STERRETT	STCTR	118	345BA107	########	########	########	nemisys		110993
47	WHI-A46-(	0	12	40413	ACTIVE	134	WHITTEM(	WHIT	174E	199EA103	########	########	########	nemisys		113307
48	BIOI2-151-	0	20	112836	UNDEFINE	120	BIOINFOR	BIOI2	162	TP08C	########	########	########	nemisys		115208
49	WHIT-EA-(	0	33	121862	UNDEFINE	134	WHITTEM(	WHIT	300C	199EA131	########	########	########	nemisys		113881
50	STCTR-11C	0	6	121257	ACTIVE	242	STERRETT	STCTR	241	TP01A	########	########	########	nemisys		115374



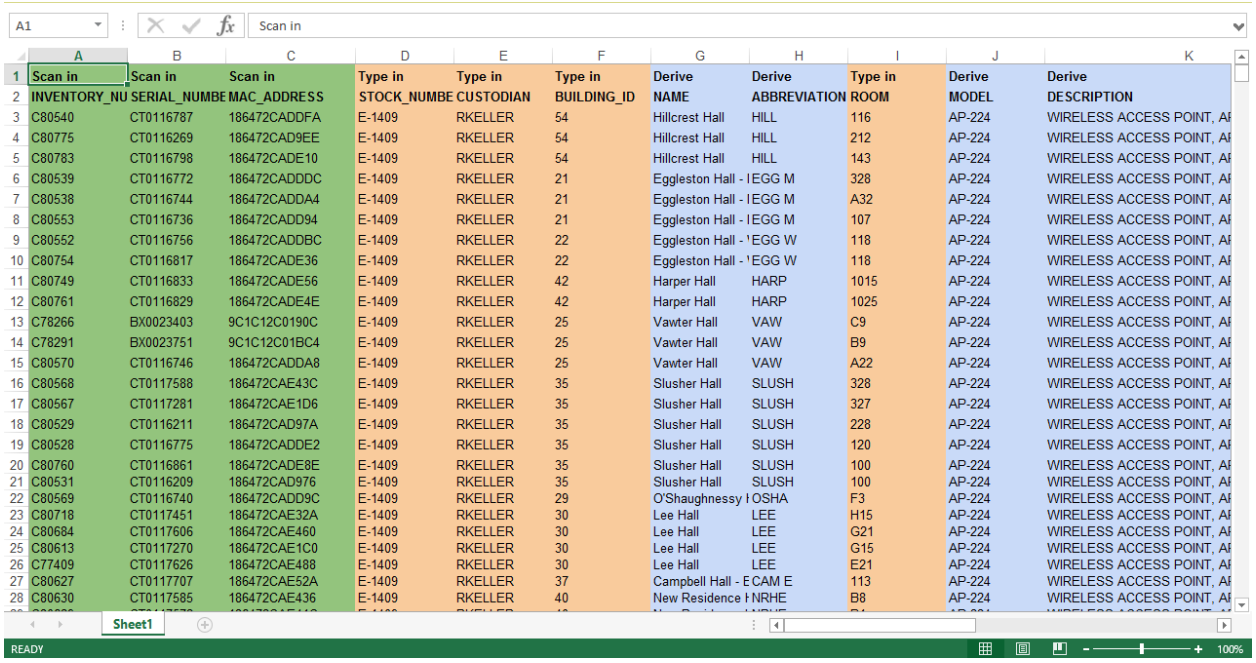
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
50	STCTR-110	0	6	121257	ACTIVE	242	STERRETT	STCTR	241	TP01A	########	########	########	nemisys		115374
51	STCTR-201	0	12	77661	UNDEFINE	242	STERRETT	STCTR	61C	345BA113	########	########	########	nemisys		110983
52	STCTR-201	0	2	93211	UNDEFINE	242	STERRETT	STCTR	61D	345BA112	########	########	########	nemisys		110984
53	STCTR-201	0	7	77652	UNDEFINE	242	STERRETT	STCTR	61A	345BA114	########	########	########	nemisys		110985
54	STCTR-201	0	9	77658	UNDEFINE	242	STERRETT	STCTR	61B	345BA112	########	########	########	nemisys		110986
55	STCTR-201	0	21	90425	UNDEFINE	242	STERRETT	STCTR	61	345BA113	########	########	########	nemisys		110994
56	WHIT-BA-	0	13	85166	PENDING	134	WHITTEM	WHIT	140B	199BA105	########	########	########	nemisys		111322
57	BIOI2-151-	0	7	91573	UNDEFINE	120	BIOINFOR	BIOI2	162	TP06B	########	########	########	nemisys		115209
58	PRIT-D18-	0	10	99677	UNDEFINE	31	PRITCHAR	PRIT	L2A	TP02B	########	########	########	nemisys		113783
59	WHI-A8-0	0	4	46130	ACTIVE	134	WHITTEM	WHIT	146	199BA109	########	########	########	nemisys		113009
60	WHI-A8-0	0	4	46130	ACTIVE	134	WHITTEM	WHIT	146	TP01B	########	########	########	nemisys		113010
61	WHI-A8-0	0	6	59700	ACTIVE	134	WHITTEM	WHIT	140B	TP02B	########	########	########	nemisys		112978
62	WHI-A8-0	0	7	59701	ACTIVE	134	WHITTEM	WHIT	144	199BA108	########	########	########	nemisys		112979
63	BIOI2-177-	0	13	91603	UNDEFINE	120	BIOINFOR	BIOI2	172	TP07B	########	########	########	nemisys		115210
64	STCTR-110	0	3	121249	PENDING	242	STERRETT	STCTR	237	TP01A	########	########	########	nemisys		115361
65	WRGHT-3	0	4	32235	ACTIVE	276	WRIGHT H	WRGHT		387AA105	########	########	########	nemisys		111953
66	WRGHT-3	0	4	80707	ACTIVE	276	WRIGHT H	WRGHT		387AA101	########	########	########	nemisys		111959
67	WHI-A8-0	0	7	59701	ACTIVE	134	WHITTEM	WHIT	144	TP02A	########	########	########	nemisys		112980
68	BIOI2-177-	0	14	91604	UNDEFINE	120	BIOINFOR	BIOI2	172	TP08C	########	########	########	nemisys		115211
69	MIL-BA-02	0	29	114914	UNDEFINE	203	MILITARY	MIL	105	274BA109	########	########	########	nemisys		115212
70	WHI-A8-0	0	1	69403	ACTIVE	134	WHITTEM	WHIT	151	199BA109	########	########	########	nemisys		113011
71	WHI-A8-0	0	1	69403	ACTIVE	134	WHITTEM	WHIT	151	TP05B	########	########	########	nemisys		113012
72	WHIT-EA-	0	1	121888	ACTIVE	134	WHITTEM	WHIT	290	199EA133	########	########	########	nemisys		114051
73	WRGHT-A	0	16	80706	ACTIVE	276	WRIGHT H	WRGHT	1	TP08B	########	########	########	nemisys		111960
74	WRGHT-A	0	5	32232	PENDING	276	WRIGHT H	WRGHT	1	TP01B	########	########	########	nemisys		111881



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
75	WRGHT-A	0	5	32232	PENDING	276	WRIGHT H	WRGHT		387AA100	########	########	########	nemisys		111882
76	WRGHT-A	0	6	32233	PENDING	276	WRIGHT H	WRGHT	1	TP06B	########	########	########	nemisys		111883
77	WRGHT-A	0	6	32233	PENDING	276	WRIGHT H	WRGHT		387AA101	########	########	########	nemisys		111884
78	WRGHT-A	0	18	72621	ACTIVE	276	WRIGHT H	WRGHT	106	TP03B	########	########	########	nemisys		111879
79	WRGHT-A	0	18	72621	ACTIVE	276	WRIGHT H	WRGHT		387AA102	########	########	########	nemisys		111880
80	WHI-A8-0	0	7	100269	ACTIVE	134	WHITTEM	WHIT	154	199EA100	########	########	########	nemisys		112811
81	WHIT-BA-	0	21	85177	PENDING	134	WHITTEM	WHIT	140B	199BA104	########	########	########	nemisys		111292
82	GRNDS-10	0	10	97936	UNDEFINE	241	GROUNDS	GRNDS	125	342AA106	########	########	########	nemisys		114989
83	GRNDS-10	0	15	72532	UNDEFINE	241	GROUNDS	GRNDS	125	342AA106	########	########	########	nemisys		114988
84	WHI-A8-0	0	14	87788	ACTIVE	134	WHITTEM	WHIT	138	TP08D	########	########	########	nemisys		113016
85	WHI-A8-0	0	13	114334	ACTIVE	134	WHITTEM	WHIT	161	199EA102	########	########	########	nemisys		112761
86	WRGHT-A	0	7	32234	PENDING	276	WRIGHT H	WRGHT	2	TP01A	########	########	########	nemisys		111885
87	MIL-BA-02	0	30	114913	UNDEFINE	203	MILITARY	MIL	105	274BA109	########	########	########	nemisys		115213
88	MIL-BA-02	0	7	81537	UNDEFINE	203	MILITARY	MIL	101	274BA100	########	########	########	nemisys		115214
89	STCTR-110	0	5	121254	PENDING	242	STERRETT	STCTR	231	TP01A	########	########	########	nemisys		115364
90	HUT-212I-	0	17	44496	UNDEFINE	103	HUTCHES	HUTCH	220	TP01A	########	########	########	nemisys		111707
91	WRGHT-A	0	3	32228	ACTIVE	276	WRIGHT H	WRGHT	109	TP02B	########	########	########	nemisys		111966
92	WRGHT-A	0	3	32228	ACTIVE	276	WRIGHT H	WRGHT		387AA103	########	########	########	nemisys		111967
93	WRGHT-3	0	5	32224	PENDING	276	WRIGHT H	WRGHT	101	387AA101	########	########	########	nemisys		111831
94	WHI-A8-0	0	2	69404	ACTIVE	134	WHITTEM	WHIT	151	TP06A	########	########	########	nemisys		113018
95	PAM-DA-0	0	16	48854	ACTIVE	153	PAMPLIN	PAM	3103	222DA128	########	########	########	nemisys		115560
96	PAM-DA-0	0	1	107675	ACTIVE	153	PAMPLIN	PAM		222DA115	########	########	########	nemisys		109577
97	WRGHT-A	0	6	32233	ACTIVE	276	WRIGHT H	WRGHT	1	TP06B	########	########	########	nemisys		111970
98	WRGHT-A	0	6	32233	ACTIVE	276	WRIGHT H	WRGHT		387AA101	########	########	########	nemisys		111971
99	WRGHT-3	0	5	32224	PENDING	276	WRIGHT H	WRGHT	101	TP02B	########	########	########	nemisys		111832

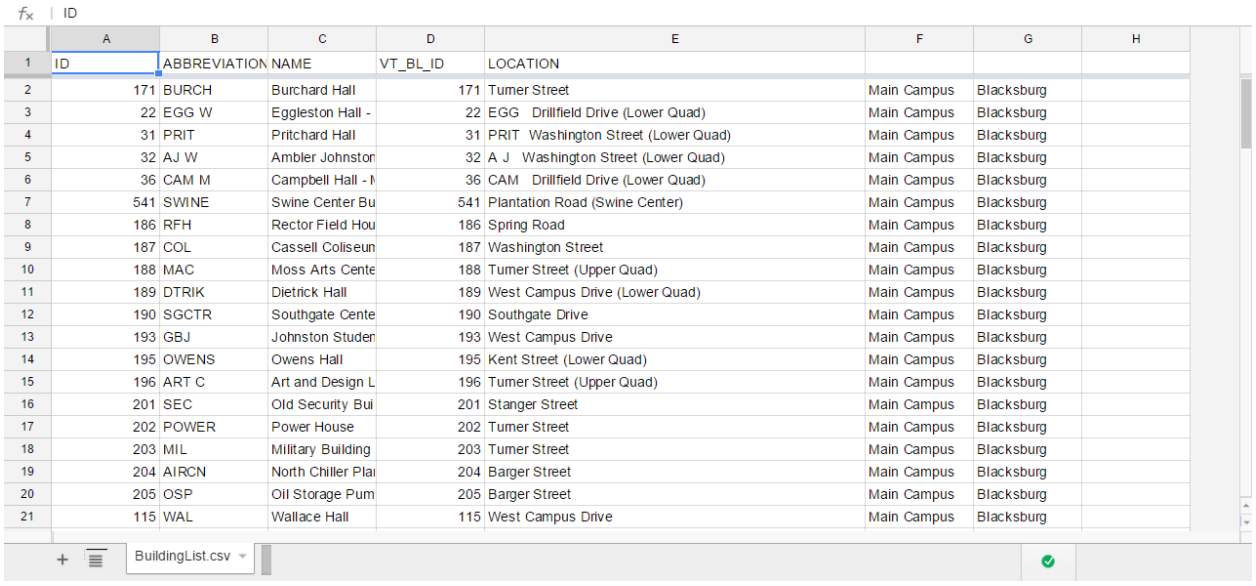
Appendix D: Sample Databases

ArubaWAPS sample data given by client - for us to understand required data.



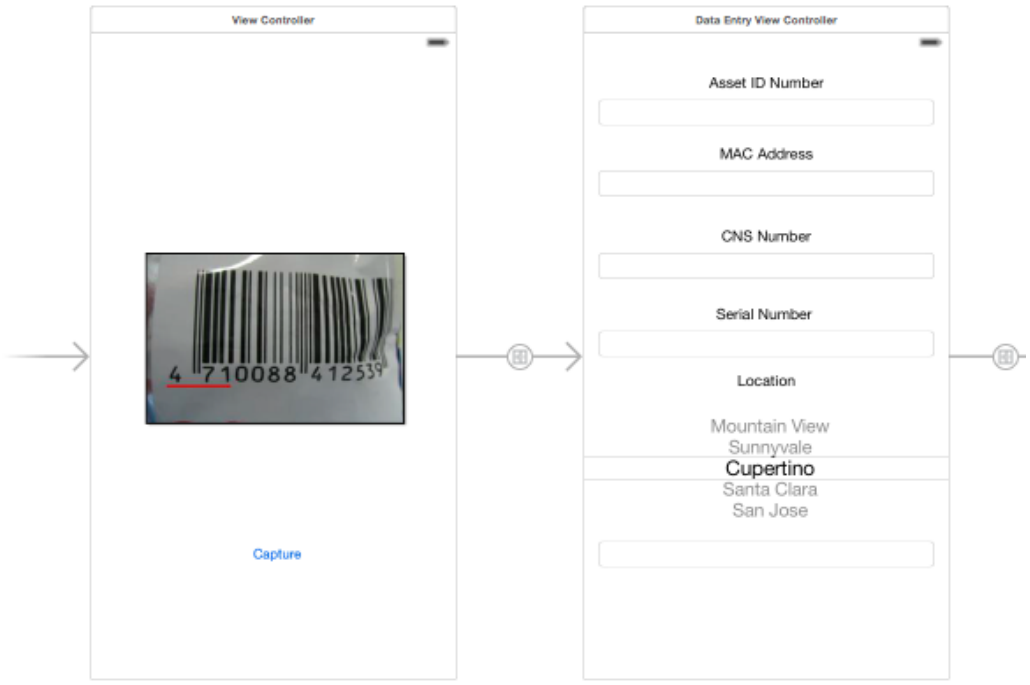
	A	B	C	D	E	F	G	H	I	J	K
1	Scan in	Scan in	Scan in	Type in	Type in	Type in	Derive	Derive	Type in	Derive	Derive
2	INVENTORY_NU	SERIAL_NUMBE	MAC_ADDRESS	STOCK_NUMBE	CUSTODIAN	BUILDING_ID	NAME	ABBREVIATION	ROOM	MODEL	DESCRIPTION
3	C80540	CT0116787	186472CADDFA	E-1409	RKELLER	54	Hillcrest Hall	HILL	116	AP-224	WIRELESS ACCESS POINT, AI
4	C80775	CT0116269	186472CAD9EE	E-1409	RKELLER	54	Hillcrest Hall	HILL	212	AP-224	WIRELESS ACCESS POINT, AI
5	C80783	CT0116798	186472CADE10	E-1409	RKELLER	54	Hillcrest Hall	HILL	143	AP-224	WIRELESS ACCESS POINT, AI
6	C80539	CT0116772	186472CADDDC	E-1409	RKELLER	21	Eggleston Hall - I	EGG M	328	AP-224	WIRELESS ACCESS POINT, AI
7	C80538	CT0116744	186472CADDA4	E-1409	RKELLER	21	Eggleston Hall - I	EGG M	A32	AP-224	WIRELESS ACCESS POINT, AI
8	C80553	CT0116736	186472CADD94	E-1409	RKELLER	21	Eggleston Hall - I	EGG M	107	AP-224	WIRELESS ACCESS POINT, AI
9	C80552	CT0116756	186472CADDBC	E-1409	RKELLER	22	Eggleston Hall - \	EGG W	118	AP-224	WIRELESS ACCESS POINT, AI
10	C80754	CT0116817	186472CADE36	E-1409	RKELLER	22	Eggleston Hall - \	EGG W	118	AP-224	WIRELESS ACCESS POINT, AI
11	C80749	CT0116833	186472CADE56	E-1409	RKELLER	42	Harper Hall	HARP	1015	AP-224	WIRELESS ACCESS POINT, AI
12	C80761	CT0116829	186472CADE4E	E-1409	RKELLER	42	Harper Hall	HARP	1025	AP-224	WIRELESS ACCESS POINT, AI
13	C78266	BX0023403	9C1C12C0190C	E-1409	RKELLER	25	Vawter Hall	VAW	C9	AP-224	WIRELESS ACCESS POINT, AI
14	C78291	BX0023751	9C1C12C01BC4	E-1409	RKELLER	25	Vawter Hall	VAW	B9	AP-224	WIRELESS ACCESS POINT, AI
15	C80570	CT0116746	186472CADDA8	E-1409	RKELLER	25	Vawter Hall	VAW	A22	AP-224	WIRELESS ACCESS POINT, AI
16	C80568	CT0117588	186472CAE43C	E-1409	RKELLER	35	Slusher Hall	SLUSH	328	AP-224	WIRELESS ACCESS POINT, AI
17	C80567	CT0117281	186472CAE1D6	E-1409	RKELLER	35	Slusher Hall	SLUSH	327	AP-224	WIRELESS ACCESS POINT, AI
18	C80529	CT0116211	186472CAD97A	E-1409	RKELLER	35	Slusher Hall	SLUSH	228	AP-224	WIRELESS ACCESS POINT, AI
19	C80528	CT0116775	186472CADDE2	E-1409	RKELLER	35	Slusher Hall	SLUSH	120	AP-224	WIRELESS ACCESS POINT, AI
20	C80760	CT0116861	186472CADE8E	E-1409	RKELLER	35	Slusher Hall	SLUSH	100	AP-224	WIRELESS ACCESS POINT, AI
21	C80531	CT0116209	186472CAD976	E-1409	RKELLER	35	Slusher Hall	SLUSH	100	AP-224	WIRELESS ACCESS POINT, AI
22	C80569	CT0116740	186472CADD9C	E-1409	RKELLER	29	O'Shaughnessy I	OSHA	F3	AP-224	WIRELESS ACCESS POINT, AI
23	C80718	CT0117451	186472CAE32A	E-1409	RKELLER	30	Lee Hall	LEE	H15	AP-224	WIRELESS ACCESS POINT, AI
24	C80684	CT0117606	186472CAE460	E-1409	RKELLER	30	Lee Hall	LEE	G21	AP-224	WIRELESS ACCESS POINT, AI
25	C80613	CT0117270	186472CAE1C0	E-1409	RKELLER	30	Lee Hall	LEE	G15	AP-224	WIRELESS ACCESS POINT, AI
26	C77409	CT0117626	186472CAE488	E-1409	RKELLER	30	Lee Hall	LEE	E21	AP-224	WIRELESS ACCESS POINT, AI
27	C80627	CT0117707	186472CAE52A	E-1409	RKELLER	37	Campbell Hall - E	CAM E	113	AP-224	WIRELESS ACCESS POINT, AI
28	C80630	CT0117585	186472CAE436	E-1409	RKELLER	40	New Residence I	NRHE	B8	AP-224	WIRELESS ACCESS POINT, AI

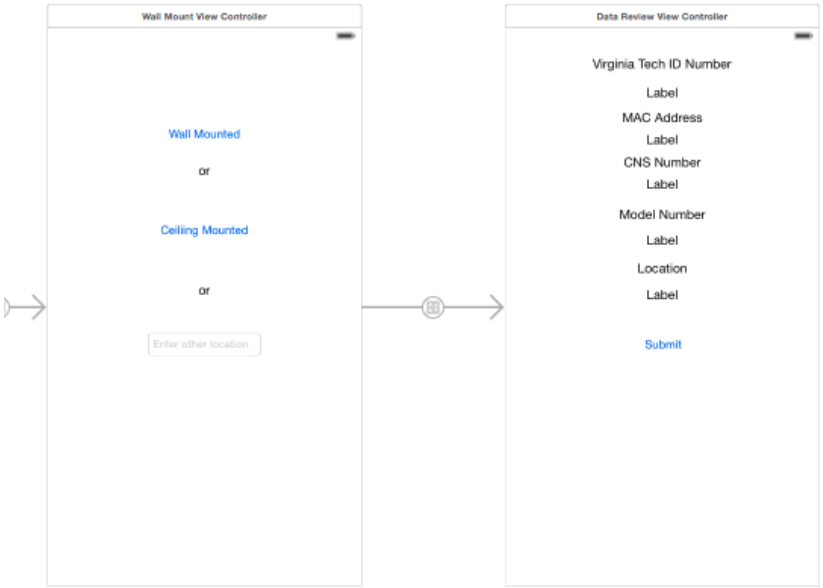
Building Abbreviations for User Accessibility

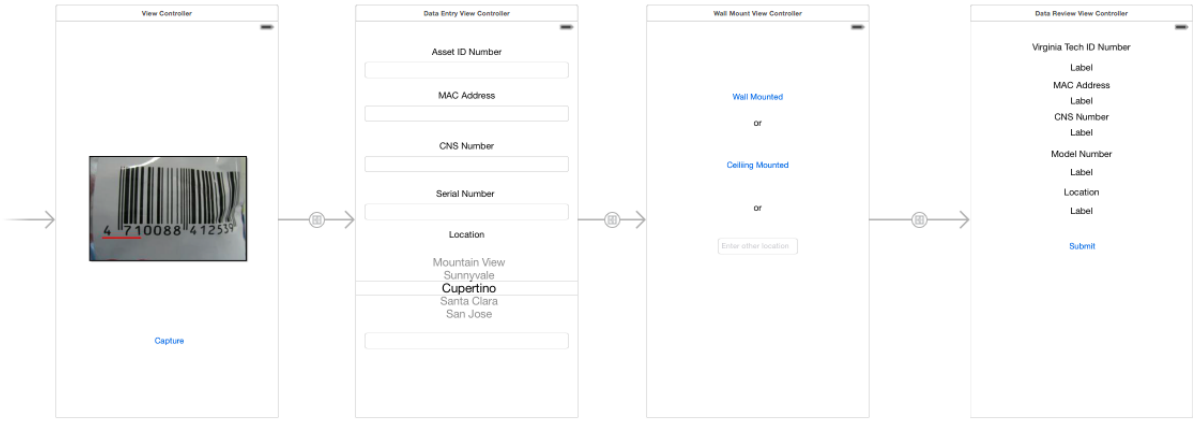


	A	B	C	D	E	F	G	H
1	ID	ABBREVIATION	NAME	VT_BL_ID	LOCATION			
2	171	BURCH	Burchard Hall	171	Turner Street	Main Campus	Blacksburg	
3	22	EGG W	Eggleston Hall -	22	EGG Drillfield Drive (Lower Quad)	Main Campus	Blacksburg	
4	31	PRIT	Pritchard Hall	31	PRIT Washington Street (Lower Quad)	Main Campus	Blacksburg	
5	32	AJ W	Ambler Johnston	32	A J Washington Street (Lower Quad)	Main Campus	Blacksburg	
6	36	CAM M	Campbell Hall - M	36	CAM Drillfield Drive (Lower Quad)	Main Campus	Blacksburg	
7	541	SWINE	Swine Center Bu	541	Plantation Road (Swine Center)	Main Campus	Blacksburg	
8	186	RFH	Rector Field Hou	186	Spring Road	Main Campus	Blacksburg	
9	187	COL	Cassell Coliseun	187	Washington Street	Main Campus	Blacksburg	
10	188	MAC	Moss Arts Cente	188	Turner Street (Upper Quad)	Main Campus	Blacksburg	
11	189	DTRIK	Dietrick Hall	189	West Campus Drive (Lower Quad)	Main Campus	Blacksburg	
12	190	SGCTR	Southgate Cente	190	Southgate Drive	Main Campus	Blacksburg	
13	193	GBJ	Johnston Studen	193	West Campus Drive	Main Campus	Blacksburg	
14	195	OWENS	Owens Hall	195	Kent Street (Lower Quad)	Main Campus	Blacksburg	
15	196	ART C	Art and Design L	196	Turner Street (Upper Quad)	Main Campus	Blacksburg	
16	201	SEC	Old Security Bui	201	Stanger Street	Main Campus	Blacksburg	
17	202	POWER	Power House	202	Turner Street	Main Campus	Blacksburg	
18	203	MIL	Military Building	203	Turner Street	Main Campus	Blacksburg	
19	204	AIRCN	North Chiller Plar	204	Barger Street	Main Campus	Blacksburg	
20	205	OSP	Oil Storage Pum	205	Barger Street	Main Campus	Blacksburg	
21	115	WAL	Wallace Hall	115	West Campus Drive	Main Campus	Blacksburg	

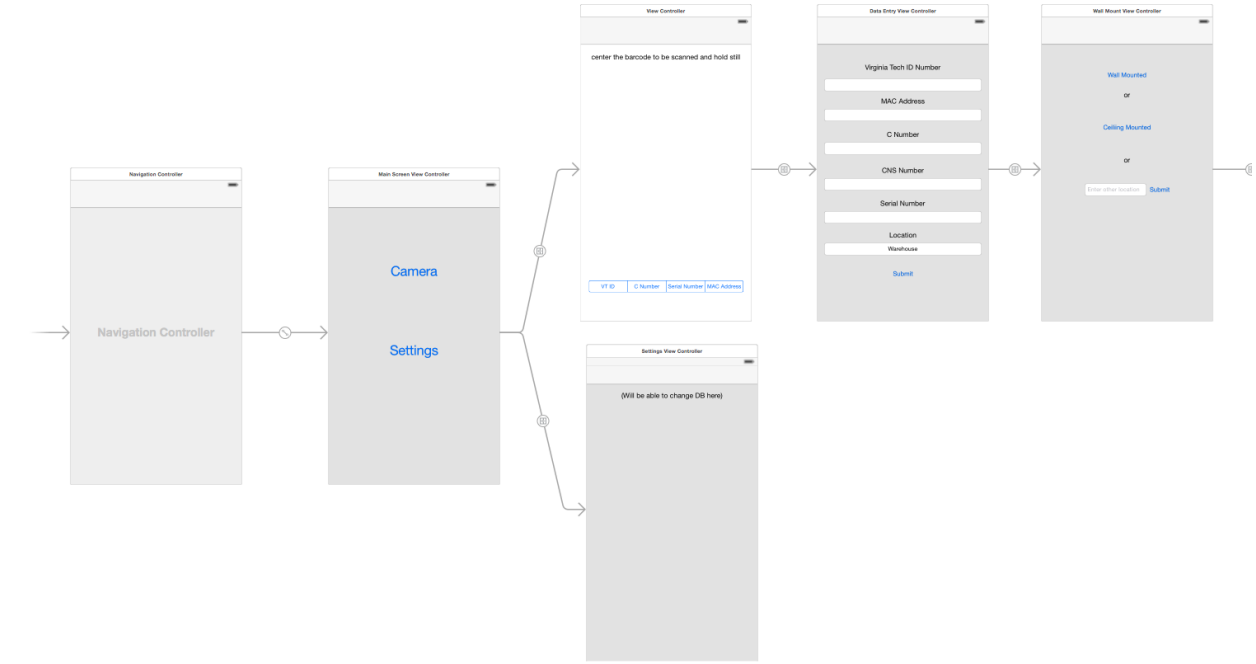
Appendix E: Prototype as of Refinement Report 2

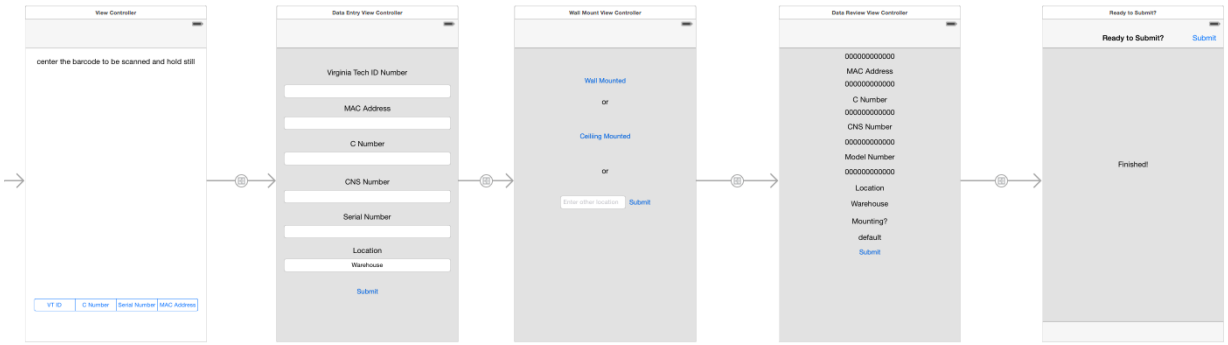





View Controller
Capture
Data Entry View Controller
Asset ID Number
MAC Address
CNS Number
Serial Number
Location
Mountain View
Sunnyvale
Cupertino
Santa Clara
San Jose
Wall Mount View Controller
Wall Mounted
or
Ceiling Mounted
or
Enter other location
Data Review View Controller
Virginia Tech ID Number
Label
MAC Address
Label
CNS Number
Label
Model Number
Label
Location
Label
Submit

Appendix F: Prototype as of Testing Report





Top Image: The first part of the logical flow for the app, including the main screen to which it diverts into the settings or the camera.
Bottom Image: The continuation of the camera's logical flow since it gets cut off after three screens in the top image.